**Section A: Basic Try-Except**

1. Write a program to divide two numbers entered by the user. Handle ZeroDivisionError using try-except.

**# Program to divide two numbers with error handling**

**try:**

**num1 = int(input("Enter the numerator:"))**

**num2 = int(input("Enter the denominator:"))**

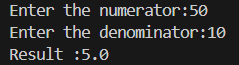
**result = num1 / num2**

**print(f"Result :{result}")**

**except ZeroDivisionError:**

**print("Error: Cannot divide by zero.")**

Output:

****

****

2. Write a program to convert a string to an integer. Handle ValueError if the input is not a valid number.

# Program to convert a string to an integer with error handling

try:

input = input("Enter a number:")

number = int(input)

print(f"Converted integer: {number}")

except ValueError:

print("Error: Input is a invalid integer.")

**Output**:





3. Accept two numbers from the user and perform addition. Use try-except to handle invalid input types.

# Program to add two numbers with error handling for invalid input

try:

num1 = float(input("Enter the first number:"))

num2 = float(input("Enter the second number:"))

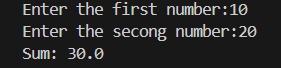
result = num1 + num2

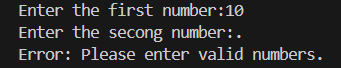
print(f"Sum: {result}")

except ValueError:

print("Error: Please enter valid numbers.")

Output:





4. Write a program to read an element from a list using an index entered by the user. Handle IndexError.

# Program to read an element from a list using user-provided index

List = [10,20,30,40,50,60,70,80,90,100]

try:

index = int(input("Enter the index (0 to 9):"))

print(f"Element at index {index} is {List[index]}")

except IndexError:

print("Error: Index is out of range.")

except ValueError:

print("Error: Please enter a valid integer index.")

Output:







**Section B: Try-Except-Else**

5. Create a program that accepts a number from the user and prints its square. Use try-except-else to handle ValueError and ensure successful computation is shown only if there's no error.

# Program to compute the square of a number with error handling

try:

num = float(input("Enter a number:"))

except ValueError:

print("Error: Please enter a valid number.")

else:

square = num\*num

print(f"Square of {num} is {square}")

Output:





6. Write a program to open a file and read contents. Use try-except-else to handle FileNotFoundError.

# Program to open a file and read its contents with error handling

try:

filename = input("Enter file name:")

file = open(filename ,'r')

except FileNotFoundError:

print("Error: File not found.")

else:

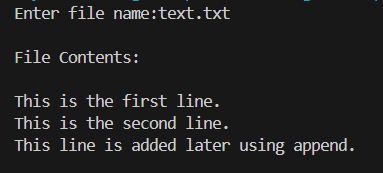
content = file.read()

print("\nFile Contents:\n")

print(content)

file.close()

Output:





7. Write a Python program to convert a number to its binary format. Use try-except-else to handle any invalid input.

# Program to convert a number to binary with error handling

try:

num = int(input("Enter a number:"))

except ValueError:

print("Error: Enter a valid integer.")

else:

binary = bin(num)

print(f"Binary of {num} is {binary}")

Output:





**Section C: Try-Finally**

8. Write a program that opens a file and ensures it gets closed, whether or not an exception occurs. Use try-finally.

# Program to open a file and ensure it is closed using try-finally

filename = input("Enter the filename: ")

try:

file = open(filename, 'r')

print("\nFile contents:\n")

print(file.read())

except FileNotFoundError:

print("Error: File not found.")

finally:

try:

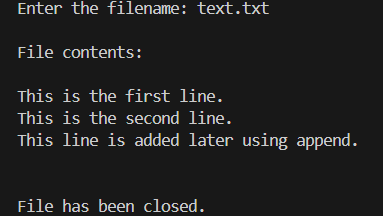
file.close()

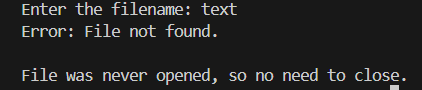
print("\nFile has been closed.")

except NameError:

print("\nFile was never opened, so no need to close.")

Output:





9. Simulate a login process where the user input is handled in a try block and a log message is printed in finally regardless of success or failure.

# Simulated login with try-except-finally

correct\_username = "admin"

correct\_password = "1234"

try:

username = input("Enter username: ")

password = input("Enter password: ")

if username == correct\_username and password == correct\_password:

print("Login successful!")

else:

print("Login failed. Incorrect credentials.")

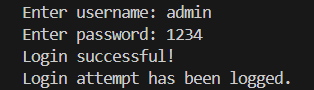
except Exception as e:

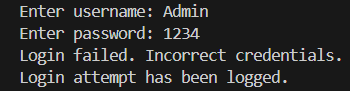
print(f"An error occurred: {e}")

finally:

print("Login attempt has been logged.")

Output:





10. Write a program that divides two numbers, catching errors with try-except, and printing a clean-up message using finally.

# Program to divide two numbers with error handling and clean-up message

try:

num1 = float(input("Enter numerator: "))

num2 = float(input("Enter denominator: "))

result = num1 / num2

print(f"Result: {result}")

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

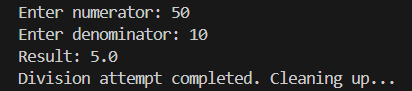
except ValueError:

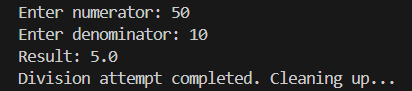
print("Error: Invalid input. Please enter numbers only.")

finally:

print("Division attempt completed. Cleaning up...")

Output:





**Section D: Combined Exception Handling**

11. Create a program that handles multiple exceptions: ZeroDivisionError, ValueError, and always prints "Execution complete" using finally.

# Program to handle multiple exceptions and always print a final message

try:

num1 = float(input("Enter numerator: "))

num2 = float(input("Enter denominator: "))

result = num1 / num2

print(f"Result: {result}")

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

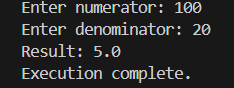
except ValueError:

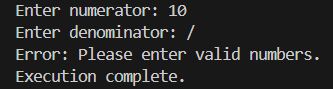
print("Error: Please enter valid numbers.")

finally:

print("Execution complete.")

Output:





12. Write a program to simulate bank withdrawal. Use try-except-else-finally to handle incorrect amount input, and always print a message whether the transaction succeeded or failed.

# Simulated bank withdrawal with full exception handling

balance = 1000.0 # Starting balance

try:

amount = float(input("Enter amount to withdraw: "))

if amount <= 0:

print("Error: Withdrawal amount must be greater than zero.")

elif amount > balance:

print("Error: Insufficient funds.")

else:

balance -= amount

print(f"Withdrawal successful! Remaining balance: ₹{balance:.2f}")

except ValueError:

print("Error: Invalid input. Please enter a numeric amount.")

Output:

