**Task No. 1:** Safe Calculator Operation

Write a Python program for a basic calculator that takes two numbers and an operation as input. Implement exception handling to ensure that division by zero is handled gracefully. If the user attempts to divide by zero, print an error message and prompt for new inputs.

**Solution:**

def safe\_calculator(num1, num2, operation):

try:

if operation == "+":

return num1 + num2

elif operation == "-":

return num1 - num2

elif operation == "\*":

return num1 \* num2

elif operation == "/":

# Check for division by zero

if num2 == 0:

raise ZeroDivisionError("Division by zero is not allowed.")

return num1 / num2

else:

raise ValueError("Invalid operation")

except ZeroDivisionError as e:

print(f"Error: {e}")

return None

except ValueError as e:

print(f"Error: {e}")

return None

result = safe\_calculator(10, 5, "+")

print(f"10 + 5 = {result}")

result = safe\_calculator(10, 2, "\*")

print(f"10 \* 2 = {result}")

result = safe\_calculator(10, 0, "/")

print(f"10 / 0 = {result}")

result = safe\_calculator(10, 5, "x") # Use "\*" instead of "x"

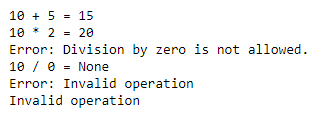
if result is not None:

print(f"10 x 5 = {result}")

else:

print("Invalid operation")

**Output:**



**Task No. 2:** File Copy with Retry

Create a Python script that copies the contents of one file to another. Implement a retry mechanism in case the file reading or writing fails due to a temporary issue (e.g., file in use by another process). Allow the user to specify the number of retry attempts.

**Solution:**

import shutil

import time

def copy\_file\_with\_retry(source\_path, destination\_path, max\_retries=3, delay\_seconds=1):

for attempt in range(1, max\_retries + 1):

try:

shutil.copy(source\_path, destination\_path)

print(f"File copied successfully from '{source\_path}' to '{destination\_path}'.")

return # If successful, exit the loop and function

except IOError as e:

print(f"Error: {e}")

if attempt < max\_retries:

print(f"Retrying in {delay\_seconds} seconds (Attempt {attempt}/{max\_retries})...")

time.sleep(delay\_seconds)

else:

print(f"Maximum retries reached. Copy operation failed.")

break

if \_\_name\_\_ == "\_\_main\_\_":

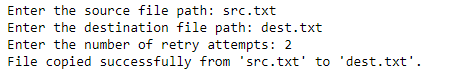
source\_file\_path = input("Enter the source file path: ")

destination\_file\_path = input("Enter the destination file path: ")

retry\_attempts = int(input("Enter the number of retry attempts: "))

copy\_file\_with\_retry(source\_file\_path, destination\_file\_path, retry\_attempts)

**Output:**



**Task No. 3:** Web Scraper with Error Handling

Develop a simple web scraper using a library like BeautifulSoup in Python. Implement exception handling to deal with common issues such as network errors, missing HTML elements, or changes in the website structure. Log relevant error messages for debugging purposes.

**Solution:**

import requests

from bs4 import BeautifulSoup

import logging

logging.basicConfig(filename='web\_scraper.log', level=logging.ERROR, format='%(asctime)s - %(levelname)s - %(message)s')

def scrape\_website(url):

try:

response = requests.get(url)

response.raise\_for\_status() # Raise an exception for HTTP errors

soup = BeautifulSoup(response.text, 'html.parser')

title = soup.title.text

print(f"Title: {title}")

except requests.exceptions.RequestException as req\_err:

logging.error(f"RequestException: {req\_err}")

print("Error: Unable to make the HTTP request.")

except requests.exceptions.HTTPError as http\_err:

logging.error(f"HTTPError: {http\_err}")

print(f"HTTP Error: {http\_err}")

except requests.exceptions.ConnectionError as conn\_err:

logging.error(f"ConnectionError: {conn\_err}")

print("Error: Unable to establish a connection to the server.")

except requests.exceptions.Timeout as timeout\_err:

logging.error(f"Timeout: {timeout\_err}")

print("Error: The request timed out.")

except requests.exceptions.TooManyRedirects as redirect\_err:

logging.error(f"TooManyRedirects: {redirect\_err}")

print("Error: Too many redirects.")

except Exception as e:

logging.error(f"Unexpected error: {e}")

print("An unexpected error occurred.")

if \_\_name\_\_ == "\_\_main\_\_":

website\_url = input("Enter the URL of the website to scrape: ")

scrape\_website(website\_url)

**Output:**





**Task No. 4:** Database Connection Retry

Write a Python program that connects to a database and performs a basic query. Implement a retry mechanism for the database connection, considering scenarios like network issues or temporary unavailability of the database server. Allow the user to set the maximum number of retry attempts.

**Solution:**

import pymongo

import time

def connect\_to\_mongodb(mongo\_params, max\_retries=3, delay\_seconds=1):

output = ""

for attempt in range(1, max\_retries + 1):

try:

client = pymongo.MongoClient(\*\*mongo\_params)

db = client.get\_database()

result = db.command("ping")

output = f"MongoDB Ping Result: {result['ok']}"

client.close()

break

except pymongo.errors.ServerSelectionTimeoutError as mongo\_error:

error\_message = f"MongoDB Error: {mongo\_error}"

if attempt < max\_retries:

error\_message += f"\nRetrying in {delay\_seconds} seconds (Attempt {attempt}/{max\_retries})..."

time.sleep(delay\_seconds)

else:

error\_message += f"\nMaximum retries reached. MongoDB connection failed."

output = error\_message

return output

if \_\_name\_\_ == "\_\_main\_\_":

mongo\_params = {

"host": "127.0.0.1",

"port": 27017,

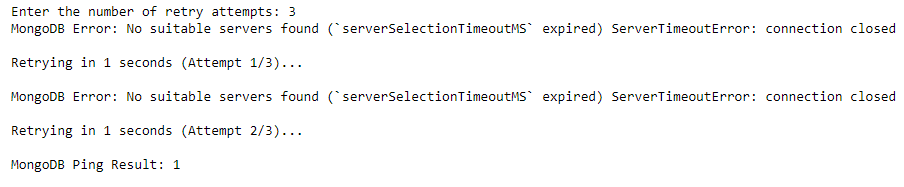
}

max\_retries = int(input("Enter the number of retry attempts: "))

result = connect\_to\_mongodb(mongo\_params, max\_retries)

print(result)

**Output:**



**Task No. 5:** Custom Exception Class

Create a Python program that simulates a banking system. Design a custom exception class, let's say InsufficientFundsError, and use it to handle situations where a user tries to withdraw more money than their account balance. Ensure that the program raises and catches this custom exception appropriately.

**Solution:**

class InsufficientFundsError(Exception):

def \_\_init\_\_(self, amount, balance):

self.amount = amount

self.balance = balance

super().\_\_init\_\_(f"Insufficient funds. Attempted to withdraw {amount}, but balance is only {balance}.")

class BankAccount:

def \_\_init\_\_(self, initial\_balance):

self.balance = initial\_balance

def withdraw(self, amount):

if amount > self.balance:

raise InsufficientFundsError(amount, self.balance)

else:

self.balance -= amount

print(f"Withdrawal successful. Remaining balance: {self.balance}")

if \_\_name\_\_ == "\_\_main\_\_":

initial\_balance = float(input("Enter the initial balance of your account: "))

account = BankAccount(initial\_balance)

try:

withdrawal\_amount = float(input("Enter the amount you want to withdraw: "))

account.withdraw(withdrawal\_amount)

except ValueError:

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

except InsufficientFundsError as e:

print(f"Error: {e}")

**Output:**

A black text on a white background

Description automatically generated