Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_12\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | 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. |
| 2 | 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. |
| 3 | 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. |
| 4 | 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. |
| 5 | 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 |

Submitted On

27-12-2023

(Date: DD/MM/YY)

**Task 1:** 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 calculator(num1,num2):

try:

result=num1/num2

return result

except ZeroDivisionError:

print("Error: It cannot divide by zero.")

return None

result=calculator(5,0)

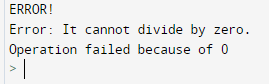
if result is not None:

print(f"The num is: {result}")

else:

print("Operation failed because of 0")

**Output:**



**Task 2:** 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 tenacity

@tenacity.retry(wait=tenacity.wait\_fixed(1), stop=tenacity.stop\_after\_attempt(3))

def copy\_file\_with\_retry(source\_file, destination\_file):

shutil.copy2(source\_file, destination\_file)

source\_file = "/content/shoaib.txt"

destination\_file = "/content/abc.txt"

retry\_attempts = 3

try:

copy\_file\_with\_retry(source\_file, destination\_file)

except tenacity.RetryError:

print(f"Failed to copy file after {retry\_attempts} attempts.")

**Task 3:** 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

def scrape\_quotes():

try:

response = requests.get('http://quotes.toscrape.com')

response.raise\_for\_status() # Raise an HTTPError for bad responses

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

quotes = soup.select('.text') # CSS selector for quote text

authors = soup.select('.author') # CSS selector for author

if len(quotes) != len(authors):

raise Exception("Number of quotes and authors don't match")

for i in range(len(quotes)):

print(f"Quote: {quotes[i].get\_text()}")

print(f"Author: {authors[i].get\_text()}")

print("")

except requests.exceptions.RequestException as e:

print(f"Error in making the request: {e}")

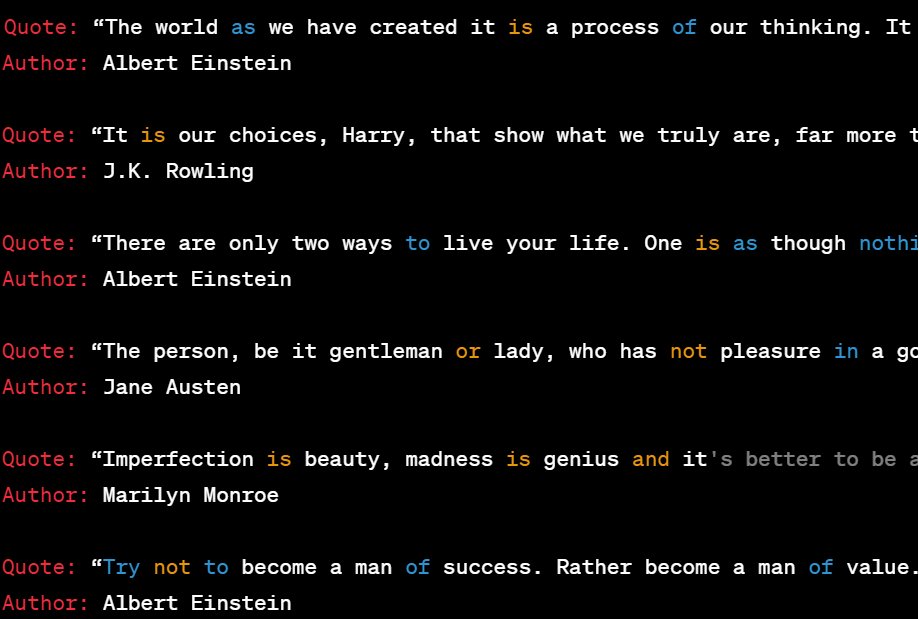
except Exception as e:

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

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

scrape\_quotes()

**Output:**

****

**Task 4:** 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 sqlite3

import time

def connect\_to\_database(database\_path, max\_retries):

retries = 0

connection = None

while retries < max\_retries:

try:

connection = sqlite3.connect(database\_path)

print("Connected to the database.")

return connection

except sqlite3.Error as e:

print(f"Error connecting to the database: {e}")

retries += 1

print(f"Retrying... (Attempt {retries}/{max\_retries})")

time.sleep(2\*\*retries) # Exponential backoff

print(f"Maximum number of retries ({max\_retries}) reached. Unable to connect to the database.")

return None

def query\_database(connection):

if connection:

try:

cursor = connection.cursor()

cursor.execute("SELECT \* FROM example\_table")

result = cursor.fetchall()

print("Query result:", result)

except sqlite3.Error as e:

print(f"Error executing query: {e}")

finally:

if connection:

connection.close()

print("Connection closed.")

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

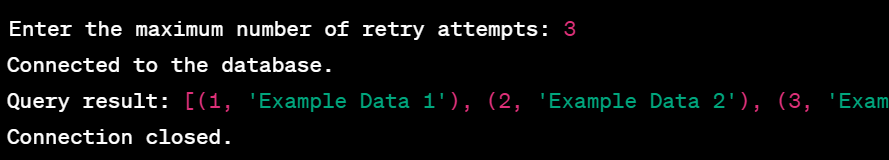
database\_path = "example.db"

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

db\_connection = connect\_to\_database(database\_path, max\_retries)

query\_database(db\_connection)

**Output:**

****

**Task 5:** 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 the balance is ${balance}.")

class BankAccount:

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

self.balance = balance

def deposit(self, amount):

self.balance += amount

print(f"Deposited ${amount}. New balance: ${self.balance}")

def withdraw(self, amount):

if amount > self.balance:

raise InsufficientFundsError(amount, self.balance)else:

self.balance -= amount

print(f"Withdrew ${amount}. New balance: ${self.balance}")try:

initial\_balance = 1000

account = BankAccount(initial\_balance)

account.deposit(500)

account.withdraw(200)

account.withdraw(1000) # This will raise an InsufficientFundsError

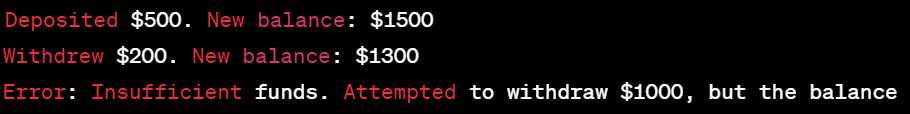
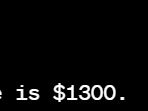
except InsufficientFundsError as e:

print(f"Error: {e}")

except Exception as e:

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

**Output:**

**** ****