**1. Exception Handling: Division Handling**

**File:** exception\_division.py

def divide\_numbers():

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.")

divide\_numbers()

**output:** Error: Cannot divide by zero.

**2. Exception Handling: File Handling Errors**

**File:** exception\_file.py

def read\_file(filename):

try:

with open(filename, 'r') as file:

content = file.read()

print(content)

except FileNotFoundError:

print("Error: File not found.")

except IOError:

print("Error: Cannot read the file.")

read\_file("sample.txt")

**output: this is a sample txt file**

**3. File Handling: Read and Write**

**File:** file\_read\_write.py

def write\_to\_file(filename, text):

with open(filename, 'w') as file:

file.write(text)

def read\_from\_file(filename):

with open(filename, 'r') as file:

print(file.read())

write\_to\_file("sample.txt", "Hello, File Handling!")

read\_from\_file("sample.txt")

output: Hello, File Handling!

**4. File Handling: CSV Processing**

**File:** csv\_processing.py

import csv

def write\_csv():

with open("data.csv", mode='w', newline='') as file:

writer = csv.writer(file)

writer.writerow(["Name", "Age", "City"])

writer.writerow(["Alice", 25, "New York"])

writer.writerow(["Bob", 30, "Los Angeles"])

def read\_csv():

with open("data.csv", mode='r') as file:

reader = csv.reader(file)

for row in reader:

print(row)

write\_csv()

read\_csv()

**output:**

['Name', 'Age', 'City']

['Alice', '25', 'New York']

['Bob', '30', 'Los Angeles']

**5. File Handling: JSON Handling**

**File:** json\_handling.py

import json

def write\_json():

data = {"name": "Mohan", "age": 25, "city": "New York"}

with open("data.json", "w") as file:

json.dump(data, file)

def read\_json():

with open("data.json", "r") as file:

data = json.load(file)

print(data)

write\_json()

read\_json()

**output:** {'name': 'Mohan', 'age': 25, 'city': 'New York'}

**6. Multi-threading: Parallel Downloads**

**File:** multithreading\_download.py

import threading

import time

def download\_file(filename):

print(f"Downloading {filename}...")

time.sleep(2)

print(f"{filename} downloaded.")

files = ["file1.txt", "file2.txt", "file3.txt"]

threads = []

for file in files:

thread = threading.Thread(target=download\_file, args=(file,))

threads.append(thread)

thread.start()

for thread in threads:

thread.join()

**output:**

Downloading file1.txt...

Downloading file2.txt...

Downloading file3.txt...

file1.txt downloaded.

file2.txt downloaded.

file3.txt downloaded.

**7. Multiprocessing: CPU-Intensive Computation**

**File:** multiprocessing\_calculations.py

import multiprocessing

def compute\_square(n):

print(f"Square of {n}: {n \* n}")

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

numbers = [1, 2, 3, 4, 5]

processes = []

for num in numbers:

process = multiprocessing.Process(target=compute\_square, args=(num,))

processes.append(process)

process.start()

for process in processes:

process.join()

**output:**

Square of 1: 1

Square of 2: 4

Square of 3: 9

Square of 5: 25

Square of 4: 16

NUMPY:

1. **WAP to basic usage of numpy**

import numpy as np

# Creating an array

arr = np.array([1, 2, 3])

print('creating array')

print(arr)

# Operations on arrays

result = arr + 2

print('\noperations array result\n',result)

# Broadcasting

**matrix = np.array([[1, 2, 3], [4, 5, 6]])**

result\_matrix = matrix \* 2

print('\nBroadcasting result matrix\n', result\_matrix)

# Linear Algebra

dot\_product = np.dot(matrix, arr)

print('\nLinear Algebra dot\_product\n', result\_matrix)

# Random numbers

random\_array = np.random.rand(3, 3)

print('\nrandom number\n', random\_array)

# Indexing and Slicing

subset = matrix[:, 1:3]

print('\nindexing slicing subset :\n',subset)

--------------------------------------------------------------------------------------------------------------------------

OUTPUT

creating array

[1 2 3]

operations array result

[3 4 5]

Broadcasting result matrix

[[ 2 4 6]

[ 8 10 12]]

Linear Algebra dot\_product

[[ 2 4 6]

[ 8 10 12]]

random number

[[0.08603371 0.82107951 0.8770816 ]

[0.43214938 0.97130027 0.83232021]

[0.21496735 0.63956808 0.10113229]]

indexing slicing subset :

[[2 3]

[5 6]]

--------------------------------------------------------------------------------------------------------------------------

**PANDAS:**

**You have a dataset of Indian cricketers with details like name, role, batting average, and number of matches played.**

import pandas as pd

print("this pandas example")

# Creating a sample dataset

data = {

    "Name": ["Virat Kohli", "Rohit Sharma", "MS Dhoni", "Sachin Tendulkar", "Rahul Dravid"],

    "Role": ["Batsman", "Batsman", "Wicketkeeper", "Batsman", "Batsman"],

    "Matches": [280, 250, 350, 463, 344],

    "Batting\_Average": [57.3, 49.0, 50.6, 44.8, 52.3]

}

# Creating a DataFrame

df = pd.DataFrame(data)

# Displaying the DataFrame

print("Cricketer Data:\n", df)

# Filtering players with Batting Average above 50

high\_avg\_players = df[df["Batting\_Average"] > 50]

print("\nPlayers with Batting Average above 50:\n", high\_avg\_players)

# Sorting by Batting Average

sorted\_df = df.sort\_values(by="Batting\_Average", ascending=False)

print("\nSorted Data by Batting Average:\n", sorted\_df)

# Calculating the average batting average

avg\_batting\_avg = df["Batting\_Average"].mean()

print("\nOverall Average Batting Average:", round(avg\_batting\_avg, 2))

**----------------------------------------------------------------------------------------------------------------------**

**OUTPUT:**

**Cricketer Data:**

**Name Role Matches Batting\_Average**

**0 Virat Kohli Batsman 280 57.3**

**1 Rohit Sharma Batsman 250 49.0**

**2 MS Dhoni Wicketkeeper 350 50.6**

**3 Sachin Tendulkar Batsman 463 44.8**

**4 Rahul Dravid Batsman 344 52.3**

**Players with Batting Average above 50:**

**Name Role Matches Batting\_Average**

**0 Virat Kohli Batsman 280 57.3**

**2 MS Dhoni Wicketkeeper 350 50.6**

**4 Rahul Dravid Batsman 344 52.3**

**Sorted Data by Batting Average:**

**Name Role Matches Batting\_Average**

**0 Virat Kohli Batsman 280 57.3**

**4 Rahul Dravid Batsman 344 52.3**

**2 MS Dhoni Wicketkeeper 350 50.6**

**1 Rohit Sharma Batsman 250 49.0**

**3 Sachin Tendulkar Batsman 463 44.8**

**Overall Average Batting Average: 50.8**

**--------------------------------------------------------------------------------------------------------------------------**

**8**

SCIPY:

**Simulate a simple population growth scenario where the rate of growth of a population is proportional to its current size.**

import numpy as np

from scipy.integrate import solve\_ivp

import matplotlib.pyplot as plt

# Step 2: Define the ODE

def population\_growth(t, P, k):

return k \* P

# Step 3: Set Initial Conditions and Solve the ODE

initial\_population = 100

growth\_constant = 0.1

solution = solve\_ivp(population\_growth, [0, 10], [initial\_population], args=(growth\_constant,), t\_eval=np.linspace(0, 10, 100))

# Step 4: Plot the Population Growth

plt.plot(solution.t, solution.y[0], label='Population Growth')

plt.xlabel('Time')

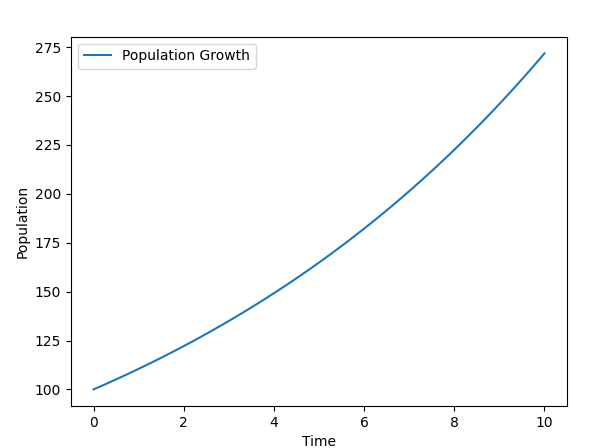
plt.ylabel('Population')

plt.legend()

plt.show()

--------------------------------------------------------------------------------------------------------------------------

**OUTPUT:**



--------------------------------------------------------------------------------------------------------------------------

**. Database Operations: SQLite CRUD Operations**

<https://sqlitebrowser.org/dl/>

Example 1: Creating a New Database and Table

import sqlite3

conn = sqlite3.connect("company.db")

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS employees (

emp\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

department TEXT

)

""")

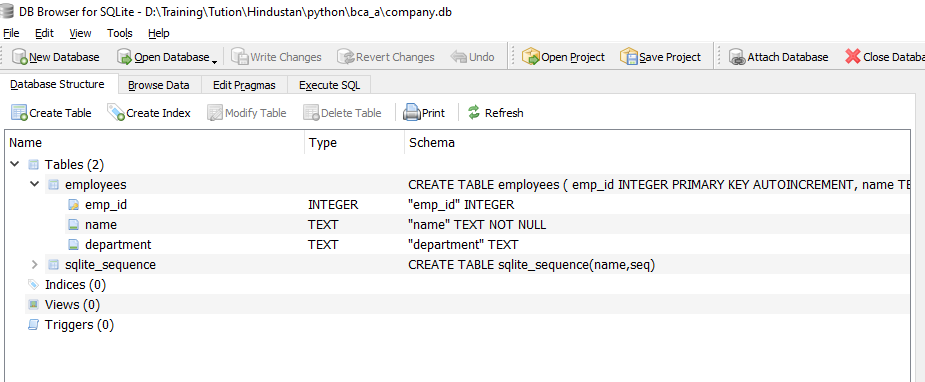
print("Table created successfully")

conn.commit()

conn.close()

------------------------------------------------------------------------------------------------------------------------------

OUTPUT:



--------------------------------------------------------------------------------------------------------------------------

**Example 2: Inserting Multiple Records**

import sqlite3

conn = sqlite3.connect("company.db")

cursor = conn.cursor()

employees = [

("Amit", "HR"),

("Pooja", "IT"),

("Raj", "Finance")

]

cursor.executemany("INSERT INTO employees (name, department) VALUES (?, ?)", employees)

conn.commit()

print("Multiple records inserted successfully")

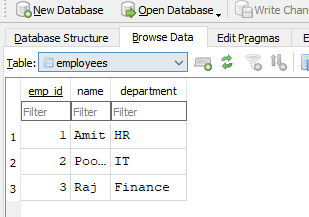
conn.close()

---------------------------------------------------------------------------------------------------------------------------

OUTPUT:

Table created successfully

Multiple records inserted successfully



---------------------------------------------------------------------------------------------------------------------------

Example 3: Retrieving and Displaying Records

import sqlite3

conn = sqlite3.connect("company.db")

cursor = conn.cursor()

cursor.execute("SELECT \* FROM employees")

rows = cursor.fetchall()

for row in rows:

print(row)

conn.close()

------------------------------------------------------------------------------------------------------------------------------

OUTPUT:

(1, 'Amit', 'HR')

(2, 'Pooja', 'IT')

(3, 'Raj', 'Finance')

--------------------------------------------------------------------------------------------------------------------------