# 1]Write a python code to find given number is Armstrong Number or

# Not. Steps:

# a. Accept Number from user in variable named as X.

# b. Print message whether number X is Armstrong or not.

# Note: Armstrong number is a number that is equal to the sum of cubes of its digits. For

# example 153. (1^3 + 5^3 + 3^3 = 153)

a = input("Enter the number : ")

sum = 0

for i in a:

ina = int(i)

cube = ina \*\* 3

sum += cube

if sum == int(a):

print(a , " is Armstrong number ")

else:

print(a , " is not a Armstrong number")

# 2] Write a code in python to count number of vowels in given string

# Steps:

# a. Accept string from user in variable named STR1.

# b. Count the number of vowels in STR1 and print.

# Eg.

# 1.STR1 = 'COCONUT' => 3

# 2.STR1 = 'CONFIDence' => 4

vowels = ['a', 'e', 'i', 'o', 'u']

str1 = input("Enter the string : ")

count = 0

for i in str1:

for a in vowels:

if i.lower() == a :

count += 1

print(f"Total vowels in the {str1} is {count}")

# 3] Write a program, which will find all such numbers between 2000 and 3000 (both included)

# such that each digit of the number is an even number. eg. 2000, 2002...2888.

for i in range(2000,3001,2):

count = 0

i = str(i)

for a in i :

a = int(a)

if a % 2 == 0:

count +=1

if count == len(i):

print(i ,end="\t")

# 4] Write a program that accepts a sentence and calculate the number of letters and

# digits. Suppose the following input is supplied to the program: hello world! 123 Then, the

# output should be: ALPHABETS 10 DIGITS 3

# (Note : Special symbols are not alphabets)

a = input("Enter the sentence")

dcount = 0

acount = 0

for i in a:

if i.isdigit():

dcount += 1

if i.isalpha():

acount += 1

print(f"The total alphabets is {acount} and digit is {dcount}")

# 5] Write a Python function that takes a list and returns a new list with unique elements of

# the first list.

# Sample List : [1,2,3,3,3,3,4,5]

# Unique List : [1, 2, 3, 4, 5]

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

uni = []

for i in a:

if i not in uni:

uni.append(i)

print(uni)

# 6] Write a Python program to make a chain of function decorators (bold, italic, underline etc.)

# in Python.

def bold\_decorator(func):

def wrapper(\*args, \*\*kwargs):

return f"<b>{func(\*args, \*\*kwargs)}</b>"

return wrapper

def italic\_decorator(func):

def wrapper(\*args, \*\*kwargs):

return f"<i>{func(\*args, \*\*kwargs)}</i>"

return wrapper

def underline\_decorator(func):

def wrapper(\*args, \*\*kwargs):

return f"<u>{func(\*args, \*\*kwargs)}</u>"

return wrapper

@bold\_decorator

@italic\_decorator

@underline\_decorator

def formatted\_text(text):

return text

# Example usage

text = "Hello, World!"

formatted\_result = formatted\_text(text)

print("Formatted Text:", formatted\_result)

# 7] Write a Python program to generate a random alphabetical character, alphabetical string

# and alphabetical string of a fixed length. Use random.choice()

import random

import string

rndchoice = random.choice(string.ascii\_letters)

print("Random alphabetical character:", rndchoice)

random\_length = random.randint(5, 15)

random\_string = ''

for \_ in range(random\_length):

random\_string += random.choice(string.ascii\_letters)

print("Random alphabetical string:", random\_string)

fixed\_length = 10

fixed\_length\_string = ''

for \_ in range(fixed\_length):

fixed\_length\_string += random.choice(string.ascii\_letters)

print("Random alphabetical string of fixed length:", fixed\_length\_string)

# 8] Create a child class Bus that will inherit all of the variables and methods of the Vehicle class

class Vehicle:

def \_\_init\_\_(self, color, max\_speed):

self.color = color

self.max\_speed = max\_speed

def display\_info(self):

print(f"Color: {self.color}, Max Speed: {self.max\_speed}")

class Bus(Vehicle):

def \_\_init\_\_(self, color, max\_speed, capacity):

# Call the constructor of the parent class (Vehicle) to initialize color and max\_speed

super().\_\_init\_\_(color, max\_speed)

self.capacity = capacity

def display\_info(self):

# Call the display\_info method of the parent class (Vehicle)

super().display\_info()

print(f"Capacity: {self.capacity}")

# Example usage

bus = Bus("Yellow", 60, 50)

bus.display\_info()

# 9] Create a Bus class that inherits from the Vehicle class. Give the capacity argument

# of Bus.seating\_capacity() a default value of 50.

class Vehicle:

def \_\_init\_\_(self, color, max\_speed):

self.color = color

self.max\_speed = max\_speed

def display\_info(self):

print(f"Color: {self.color}, Max Speed: {self.max\_speed}")

class Bus(Vehicle):

def \_\_init\_\_(self, color, max\_speed, capacity=50):

super().\_\_init\_\_(color, max\_speed)

self.capacity = capacity

def seating\_capacity(self):

print(f"Seating Capacity of the Bus: {self.capacity}")

# Example usage

bus = Bus("Yellow", 60)

bus.display\_info()

bus.seating\_capacity()

# 10] Create a Bus child class that inherits from the Vehicle class. The default fare charge of any

# vehicle is seating capacity \* 100. If Vehicle is Bus instance, we need to add an extra 10% on

# full fare as a maintenance charge. So total fare for bus instance will become the final amount =

# total fare + 10% of the total fare.

class Vehicle:

def \_\_init\_\_(self, color, max\_speed, capacity):

self.color = color

self.max\_speed = max\_speed

self.capacity = capacity

def fare\_charge(self):

fare = self.capacity \* 100

return fare

class Bus(Vehicle):

def \_\_init\_\_(self, color, max\_speed, capacity):

super().\_\_init\_\_(color, max\_speed, capacity)

def fare\_charge(self):

fare = super().fare\_charge()

if isinstance(self, Bus):

fare += fare \* 0.1 # Adding 10% maintenance charge for buses

return fare

# Example usage

bus = Bus("Yellow", 60, 50)

print("Total fare for bus:", bus.fare\_charge())

# 11] Write a Python class named Student with two attributes student\_name, marks. Modify the

# attribute values of the said class and print the original and modified values of the said

# attributes.

class Student:

def \_\_init\_\_(self, student\_name, marks):

self.student\_name = student\_name

self.marks = marks

# Create an instance of the Student class

student = Student("John", 85)

# Print the original attribute values

print("Original Student Name:", student.student\_name)

print("Original Marks:", student.marks)

# Modify the attribute values

student.student\_name = "Alice"

student.marks = 90

# Print the modified attribute values

print("Modified Student Name:", student.student\_name)

print("Modified Marks:", student.marks)

# 12] Write a Python program to match a string that contains only upper and lowercase

# letters, numbers, and underscores.

import re

def match\_string(text):

pattern = r'^[a-zA-Z0-9\_]\*$'

if re.match(pattern, text):

return True

else:

return False

# Test cases

strings\_to\_test = ["Hello\_World123", "abcDEF456", "123\_456", "special@chars", "Spaces Not Allowed"]

for text in strings\_to\_test:

if match\_string(text):

print(f"'{text}' matches the pattern.")

else:

print(f"'{text}' does not match the pattern.")

# 13] Write a python program to validate the password by using regular expression.

# a. Complexity requirement is that we need at least one capital letter, one number and one

# special character.

# b. We also need the length of the password to be between 8 and 18.

import re

def validate\_password(password):

# Ensure length is between 8 and 18 characters

if len(password) < 8 or len(password) > 18:

return False

# Ensure at least one capital letter, one number, and one special character

pattern = r'^(?=.\*[A-Z])(?=.\*\d)(?=.\*[!@#$%^&\*()\_+{}|:<>?]).+$'

if re.match(pattern, password):

return True

else:

return False

# Test cases

passwords\_to\_test = ["Abc123!@#", "Password123", "short", "Toolongpassword123456", "NoSpecialCharacter123"]

for password in passwords\_to\_test:

if validate\_password(password):

print(f"'{password}' is a valid password.")

else:

print(f"'{password}' is not a valid password.")

# 14] Write a python program to validate the URL by using regular expression. 16]

import re

def validate\_url(url):

# Regular expression pattern for URL validation

pattern = r'^(https?|ftp)://[^\s/$.?#].[^\s]\*$'

if re.match(pattern, url):

return True

else:

return False

# Test cases

urls\_to\_test = [

"http://www.example.com",

"https://example.com/page",

"ftp://ftp.example.com/file",

"invalid-url",

"http://www.invalid url.com",

"https://www.invalid?url.com"

]

for url in urls\_to\_test:

if validate\_url(url):

print(f"'{url}' is a valid URL.")

else:

print(f"'{url}' is not a valid URL.")

# 15] Write a python program to validate an email address by using regular expression.

import re

def validate\_email(email):

# Regular expression pattern for email validation

pattern = r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

if re.match(pattern, email):

return True

else:

return False

# Test cases

emails\_to\_test = [

"example@example.com",

"user123@example.co.uk",

"user.name@example.org",

"invalid-email",

"invalid@.com",

"@example.com",

"user@examplecom"

]

for email in emails\_to\_test:

if validate\_email(email):

print(f"'{email}' is a valid email address.")

else:

print(f"'{email}' is not a valid email address.")

# 16] Write a python program which consists of - try, except, else, finally blocks.

def divide(x, y):

try:

result = x / y

except ZeroDivisionError:

print("Error: Division by zero!")

else:

print("Division successful.")

print("Result:", result)

finally:

print("Executing 'finally' block.")

# Test cases

print("Test Case 1:")

divide(10, 2)

print("\nTest Case 2:")

divide(10, 0)

# 17] Write a python program which raises the exception with a message.

def divide(x, y):

if y == 0:

raise ZeroDivisionError("Error: Division by zero is not allowed!")

else:

return x / y

# Test cases

try:

result = divide(10, 0)

except ZeroDivisionError as e:

print(e)

else:

print("Result:", result)

# 18] Write a Python multithreading program to print the thread name and corresponding process

# for each task (assume that there are four tasks).

import threading

import os

def task(task\_number):

print(f"Task {task\_number} is running in thread: {threading.current\_thread().name} (Process ID: {os.getpid()})")

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

# Create four threads for four tasks

threads = []

for i in range(4):

thread = threading.Thread(target=task, args=(i+1,))

threads.append(thread)

thread.start()

# Wait for all threads to finish

for thread in threads:

thread.join()

print("All tasks are completed.")

# 19] Write a Python multithreading program which creates two threads, one for calculating the

# square of a given number and other for calculating the cube of a given number.

import threading

def calculate\_square(number):

print(f"Square of {number}: {number \*\* 2}")

def calculate\_cube(number):

print(f"Cube of {number}: {number \*\* 3}")

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

number = 5

# Create two threads, one for square and one for cube

square\_thread = threading.Thread(target=calculate\_square, args=(number,))

cube\_thread = threading.Thread(target=calculate\_cube, args=(number,))

# Start both threads

square\_thread.start()

cube\_thread.start()

# Wait for both threads to finish

square\_thread.join()

cube\_thread.join()

print("Main thread finished.")

# 20] Given a file called myfile.txt which contains the text: “Python is object oriented programming

# language”. Write a program in Python that transforms the content of the file by writing each word

# in a separate line.

# Open the input file in read mode

with open("myfile.txt", "r") as file:

# Read the content of the file

content = file.read()

# Split the content into words

words = content.split()

# Open a new file in write mode

with open("transformed\_file.txt", "w") as new\_file:

# Write each word on a separate line

for word in words:

new\_file.write(word + "\n")

print("Transformation complete. Check transformed\_file.txt for the result.")

# 21] Write a Python program that displays the longest word found in a text file

def longest\_word(filename):

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

# Read the content of the file

content = file.read()

# Split the content into words

words = content.split()

# Find the longest word

longest = max(words, key=len)

return longest

# Test the function with a file named "example.txt"

filename = "example.txt" # Replace "example.txt" with the name of your text file

longest = longest\_word(filename)

print("Longest word in the file:", longest)

# 22] Write a function in python that allows you to count the frequency of repetition of each word

# found in a given file.

def count\_word\_frequency(filename):

word\_frequency = {}

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

# Read the content of the file

content = file.read()

# Split the content into words

words = content.split()

# Count the frequency of each word

for word in words:

if word in word\_frequency:

word\_frequency[word] += 1

else:

word\_frequency[word] = 1

return word\_frequency

# Test the function with a file named "example.txt"

filename = "example.txt" # Replace "example.txt" with the name of your text file

word\_frequency = count\_word\_frequency(filename)

# Print the frequency of each word

for word, frequency in word\_frequency.items():

print(f"'{word}': {frequency}")

# 23] Write a Python program which allows you to extract the content of a file from the 3rd line to

# the 7th line and save it in another file called extract\_content.txt.

def extract\_content(input\_file, output\_file):

with open(input\_file, 'r') as file:

# Read all lines from the input file

lines = file.readlines()

# Extract lines from 3rd to 7th

extracted\_lines = lines[2:7]

# Write extracted lines to the output file

with open(output\_file, 'w') as file:

file.writelines(extracted\_lines)

# Test the function with a file named "input\_file.txt"

input\_file = "input\_file.txt" # Replace "input\_file.txt" with the name of your input file

output\_file = "extract\_content.txt"

extract\_content(input\_file, output\_file)

print(f"Content from {input\_file} extracted and saved in {output\_file}.")

# 24] Create the following DataFrame Sales containing year wise sales figures for five

# salespersons in INR. Use the years as column labels, and salesperson names as row labels.

# 2018 2019 2020 2021

# Kapil 110 205 177 189 Kamini 130 165 175 190 Shikhar 115 206 157 179 Mohini 118 198

# 183 169

# 1. Create the DataFrame.

# 2. Display the row labels of Sales.

# 3. Display the column labels of Sales.

# 4. Display the data types of each column of Sales.

# 5. Display the dimensions, shape, size and values of Sales

import pandas as pd

# Create the DataFrame Sales

data = {

'2018': [110, 130, 115, 118],

'2019': [205, 165, 206, 198],

'2020': [177, 175, 157, 183],

'2021': [189, 190, 179, 169]

}

salespersons = ['Kapil', 'Kamini', 'Shikhar', 'Mohini']

sales = pd.DataFrame(data, index=salespersons)

# Display the row labels of Sales

print("Row labels of Sales:")

print(sales.index)

# Display the column labels of Sales

print("\nColumn labels of Sales:")

print(sales.columns)

# Display the data types of each column of Sales

print("\nData types of each column of Sales:")

print(sales.dtypes)

# Display the dimensions, shape, size, and values of Sales

print("\nDimensions of Sales:")

print(sales.ndim)

print("\nShape of Sales:")

print(sales.shape)

print("\nSize of Sales:")

print(sales.size)

print("\nValues of Sales:")

print(sales.values)

# 25] Plot the following data on a line chart and customize the chart according to the belowgiven instructions:

# Month January February March April May Sales 510 350 475 580 600 Weekly Sales Report

# 1. Write a title for the chart “The Monthly Sales Report“

# 2. Write the appropriate titles of both the axes

# 3. Write code to Display legends

# 4. Display blue color for the line

# 5. Use the line style – dashed

# 6. Display diamond style markers on data points

import matplotlib.pyplot as plt

# Data

months = ['January', 'February', 'March', 'April', 'May']

sales = [510, 350, 475, 580, 600]

# Plot the data

plt.plot(months, sales, color='blue', linestyle='--', marker='D', markersize=8, label='Sales')

# Customize the chart

plt.title('The Monthly Sales Report')

plt.xlabel('Month')

plt.ylabel('Sales')

plt.legend()

# Display the chart

plt.show()

# 26] Observe following data and plot data according to given instructions:

# Batsman 2017 2018 2019 2020 Virat Kohli 2501 1855 2203 1223 Steve Smith 2340 2250 2003

# 1153 Babar Azam 1750 2147 1896 1008 Rohit Sharma 1463 1985 1854 1638 Kane Williamson

# 1256 1785 1874 1974 Jos Butler 1125 1853 1769 1436

# 1. Create a bar chart to display data of Virat Kohli & Rohit Sharma.

# 2. Customize the chart in this manner

# 1. Use different widths

# 2. Use different colors to represent different years score

# 3. Display appropriate titles for axis and chart

# 4. Show legends

# 5. Create a bar chart to display data of Steve Smith, Kane Williamson & Jos Butler.

# Customize Chart as per your wish.

# 6. Display data of all players for the specific year.

import matplotlib.pyplot as plt

# Data

batsmen = ['Virat Kohli', 'Steve Smith', 'Babar Azam', 'Rohit Sharma', 'Kane Williamson', 'Jos Butler']

years = ['2017', '2018', '2019', '2020']

scores = {

'Virat Kohli': [2501, 1855, 2203, 1223],

'Steve Smith': [2340, 2250, 2003, 1153],

'Babar Azam': [1750, 2147, 1896, 1008],

'Rohit Sharma': [1463, 1985, 1854, 1638],

'Kane Williamson': [1256, 1785, 1874, 1974],

'Jos Butler': [1125, 1853, 1769, 1436]

}

# Plot for Virat Kohli & Rohit Sharma

plt.figure(figsize=(10, 6)) # Set the figure size

for i, (batsman, color) in enumerate(zip(['Virat Kohli', 'Rohit Sharma'], ['blue', 'orange'])):

plt.bar([x + i \* 0.2 for x in range(len(years))], scores[batsman], width=0.2, color=color, label=batsman)

plt.title('Virat Kohli & Rohit Sharma - Yearly Scores')

plt.xlabel('Year')

plt.ylabel('Score')

plt.xticks(range(len(years)), years)

plt.legend()

plt.show()

# Plot for Steve Smith, Kane Williamson & Jos Butler

plt.figure(figsize=(10, 6)) # Set the figure size

for i, batsman in enumerate(['Steve Smith', 'Kane Williamson', 'Jos Butler']):

plt.bar([x + i \* 0.2 for x in range(len(years))], scores[batsman], width=0.2, label=batsman)

plt.title('Steve Smith, Kane Williamson & Jos Butler - Yearly Scores')

plt.xlabel('Year')

plt.ylabel('Score')

plt.xticks(range(len(years)), years)

plt.legend()

plt.show()

# Display data of all players for the specific year (e.g., 2019)

year = '2019'

plt.figure(figsize=(10, 6)) # Set the figure size

for batsman in batsmen:

plt.bar(batsman, scores[batsman][years.index(year)], label=batsman)

plt.title(f'Yearly Scores of All Players in {year}')

plt.xlabel('Batsmen')

plt.ylabel('Score')

plt.legend()

plt.xticks(rotation=45)

plt.show()

# 27] WAP to create a 3\*3 numpy array with all the elements as per the user choice and print

# the sum of all elements of the array.

import numpy as np

# Function to create a 3x3 array with user input elements

def create\_array():

elements = []

for \_ in range(3):

row = []

for \_ in range(3):

element = float(input("Enter element for the array: "))

row.append(element)

elements.append(row)

return np.array(elements)

# Create the array

array = create\_array()

# Print the array

print("Array:")

print(array)

# Print the sum of all elements in the array

print("Sum of all elements:", np.sum(array))

# 28] WAP to perform basic arithmetic operations on 1D and 2D array .

import numpy as np

# Create 1D array

array1d = np.array([1, 2, 3, 4, 5])

# Create 2D array

array2d = np.array([[1, 2, 3], [4, 5, 6]])

# Addition

print("Addition:")

print("1D array + 5:", array1d + 5)

print("2D array + 2:")

print(array2d + 2)

# Subtraction

print("\nSubtraction:")

print("1D array - 2:", array1d - 2)

print("2D array - 1:")

print(array2d - 1)

# Multiplication

print("\nMultiplication:")

print("1D array \* 3:", array1d \* 3)

print("2D array \* 2:")

print(array2d \* 2)

# Division

print("\nDivision:")

print("1D array / 2:", array1d / 2)

print("2D array / 3:")

print(array2d / 3)

# Element-wise square root

print("\nSquare Root:")

print("Square root of 1D array:", np.sqrt(array1d))

print("Square root of 2D array:")

print(np.sqrt(array2d))

# Element-wise exponential

print("\nExponential:")

print("Exponential of 1D array:", np.exp(array1d))

print("Exponential of 2D array:")

print(np.exp(array2d))

# 29]Write a Menu Driver Program to add, display, update, delete and exit in a student database

# containing Student\_id,Student\_name,Course through Python-MongoDB connectivity.

import pymongo

# Function to connect to MongoDB

def connect\_to\_mongodb():

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["student\_database"]

collection = db["students"]

return collection

# Function to add a student record

def add\_student(collection):

student\_id = input("Enter Student ID: ")

student\_name = input("Enter Student Name: ")

course = input("Enter Course: ")

student = {"Student\_id": student\_id, "Student\_name": student\_name, "Course": course}

collection.insert\_one(student)

print("Student added successfully!")

# Function to display all student records

def display\_students(collection):

students = collection.find()

for student in students:

print(student)

# Function to update a student record

def update\_student(collection):

student\_id = input("Enter Student ID to update: ")

new\_course = input("Enter new Course: ")

collection.update\_one({"Student\_id": student\_id}, {"$set": {"Course": new\_course}})

print("Student record updated successfully!")

# Function to delete a student record

def delete\_student(collection):

student\_id = input("Enter Student ID to delete: ")

collection.delete\_one({"Student\_id": student\_id})

print("Student record deleted successfully!")

# Main function

def main():

collection = connect\_to\_mongodb()

while True:

print("\nMenu:")

print("1. Add Student")

print("2. Display Students")

print("3. Update Student")

print("4. Delete Student")

print("5. Exit")

choice = input("Enter your choice: ")

if choice == "1":

add\_student(collection)

elif choice == "2":

display\_students(collection)

elif choice == "3":

update\_student(collection)

elif choice == "4":

delete\_student(collection)

elif choice == "5":

print("Exiting the program...")

break

else:

print("Invalid choice! Please enter a valid option.")

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

main()

# 30] Demonstrate step by step MongoDB connection in Python

import pymongo

# Step 1: Establish a connection to MongoDB

def Conn():

try:

# Connect to MongoDB server (default host and port)

client = pymongo.MongoClient("mongodb://localhost:27017/")

print("Connected to MongoDB successfully!")

return client

except pymongo.errors.ConnectionFailure as e:

print("Could not connect to MongoDB:", e)

# Step 2: Connect to a specific database

def Conn\_to\_database(client, database\_name):

try:

# Access the specified database

db = client[database\_name]

print(f"Connected to database '{database\_name}' successfully!")

return db

except Exception as e:

print("Error connecting to database:", e)

# Step 3: Access a specific collection within the database

def access\_collection(db, collection\_name):

try:

# Access the specified collection within the database

collection = db[collection\_name]

print(f"Accessed collection '{collection\_name}' successfully!")

return collection

except Exception as e:

print("Error accessing collection:", e)

# Step 4: Perform CRUD operations or other tasks with the collection

def main():

# Step 1: Connect to MongoDB

client = Conn()

# Step 2: Connect to a specific database

database\_name = "my\_database"

db = Conn\_to\_database(client, database\_name)

# Step 3: Access a specific collection within the database

collection\_name = "my\_collection"

collection = access\_collection(db, collection\_name)

# Step 4: Perform CRUD operations or other tasks with the collection

# For example, you can insert documents into the collection, query documents, update documents, delete documents, etc.

# Close the MongoDB connection when done

client.close()

print("Connection to MongoDB closed.")

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

main()

# 31] Write a Menu Driver Program to add, display, search, sort and exit in book database

# containing

# Book\_id, Book\_name, Book\_author through Python-MongoDB connectivity.

import pymongo

# Function to connect to MongoDB

def Conn():

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["book\_database"]

collection = db["books"]

return collection

# Function to add a book record

def add\_book(collection):

book\_id = input("Enter Book ID: ")

book\_name = input("Enter Book Name: ")

book\_author = input("Enter Book Author: ")

book = {"Book\_id": book\_id, "Book\_name": book\_name, "Book\_author": book\_author}

collection.insert\_one(book)

print("Book added successfully!")

# Function to display all book records

def display\_books(collection):

books = collection.find()

for book in books:

print(book)

# Function to search for a book by name

def search\_book(collection):

book\_name = input("Enter Book Name to search: ")

books = collection.find({"Book\_name": book\_name})

for book in books:

print(book)

# Function to sort books by name

def sort\_books(collection):

books = collection.find().sort("Book\_name")

for book in books:

print(book)

# Main function

def main():

collection = Conn()

while True:

print("\nMenu:")

print("1. Add Book")

print("2. Display Books")

print("3. Search Book")

print("4. Sort Books")

print("5. Exit")

choice = input("Enter your choice: ")

if choice == "1":

add\_book(collection)

elif choice == "2":

display\_books(collection)

elif choice == "3":

search\_book(collection)

elif choice == "4":

sort\_books(collection)

elif choice == "5":

print("Exiting the program...")

break

else:

print("Invalid choice! Please enter a valid option.")

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

main()