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)

Answer:-

```
# Accept number from the user

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


# Calculate the number of digits in the number

num_digits = len(str(X))


# Initialize the sum variable

sum = 0


# Temporary variable to store the original number

temp = X


# Calculate the sum of cubes of each digit

while temp > 0:

    digit = temp % 10

    sum += digit ** num_digits

    temp //= 10


# Check if the number is an Armstrong number

if X == sum:

    print(X, "is an Armstrong number.")

else:

    print(X, "is not an 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

Answer:-

```
# Accept string from the user
STR1 = input("Enter a string: ")


# Convert the string to lowercase to handle both uppercase and lowercase vowels
str_lower = STR1.lower()


# Initialize the vowel count variable
count = 0


# Define a list of vowels
vowels = ['a', 'e', 'i', 'o', 'u']


# Count the number of vowels in the string
for char in str_lower:
    if char in vowels:
        count += 1


# Print the count of vowels
print("Number of vowels in the string:", 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.

Answer:-

```
# Initialize an empty list to store the numbers
even_numbers = []


# Iterate through numbers from 2000 to 3000 (both inclusive)
for num in range(2000, 3001):
    # Convert the number to a string
    num_str = str(num)


    # Flag to keep track if all digits are even
    all_even = True


    # Check if each digit is even
    for digit in num_str:
        if int(digit) % 2 != 0:
            all_even = False
            break


    # If all digits are even, add the number to the list
    if all_even:
        even_numbers.append(num)


# Print the numbers with all even digits
print("Numbers with all even digits between 2000 and 3000:")
print(even_numbers)
```

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)

Answer:-

```
# Accept a sentence from the user
sentence = input("Enter a sentence: ")

# Initialize count variables for letters and digits
letter_count = 0
digit_count = 0

# Iterate through each character in the sentence
for char in sentence:
    # Check if the character is a letter
    if char.isalpha():
        letter_count += 1
    # Check if the character is a digit
    elif char.isdigit():
        digit_count += 1

# Print the count of letters and digits
print("ALPHABETS", letter_count, "DIGITS", digit_count)
```

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]

Answer:-

```
def get_unique_elements(input_list):
    # Create an empty list to store unique elements
    unique_list = []

    # Iterate through each element in the input list
    for element in input_list:
        # Check if the element is not already in the unique list
        if element not in unique_list:
            # Add the element to the unique list
            unique_list.append(element)

    # Return the unique list
    return unique_list

# Example usage:
sample_list = [1, 2, 3, 3, 3, 3, 4, 5]
unique_list = get_unique_elements(sample_list)
print("Unique List:", unique_list)
```

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

Answer:-

```
#Q6 Write a Python program to make a chain of function decorators (bold,
#italic, underline etc.) in Python



def make_bold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def make_italic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

def make_underline(fn):
    def wrapped():
        return "<u>" + fn() + "</u>"
    return wrapped
@make_bold
@make_italic
@make_underline

def hello():
    return "hello world"
print(hello())
```

7. Write a Python program to generate a random alphabetical character, alphabetical string and alphabetical string of a fixed length. Use random.choice()

Answer:-

```
#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
print("Generate a random alphabetical character:")
print(random.choice(string.ascii_letters))
print("\nGenerate a random alphabetical string:")
max_length = 255
str1 = ""
for i in range(random.randint(1, max_length)):
   str1 += random.choice(string.ascii_letters)
print(str1)
print("\nGenerate a random alphabetical string of a fixed length:")
str1 = ""
for i in range(10):
   str1 += random.choice(string.ascii_letters)
print(str1)
```

8. Write a generator function to print the table of the given number.

Answer:-

```
def table_generator(number, limit):
    current = 1
    while current <= limit:
        yield number * current
        current += 1


# Test the generator function
number = int(input("Enter a number: "))
limit = int(input("Enter the limit: "))


table = table_generator(number, limit)


print(f"Table of {number} up to {limit}:")
for value in table:
    print(value)
```

9. Write a python program to create University package which contains result modules. Also create sub-package College which contains exam module. Create test module and add getdata() and display() and access University and College package in it. Note: Assume suitable data.

Answer:-

First, let's create the directory structure for the packages:

```
university_package/
    __init__.py
    result.py

university_package/college/
    __init__.py
    exam.py

test_module.py
```

CODE:

result.py

```
class Result:
    def get_data(self):
        # Assume code to fetch result data from a database or file
        result_data = ...  # Fetch the result data
        return result_data

    def display(self, result_data):
        # Assume code to display the result data
        print("Result Data:")
         print(result_data)
```

exam.py

```
class Exam:
    def get_data(self):
        # Assume code to fetch exam data from a database or file
        exam_data = ...  # Fetch the exam data
        return exam_data

    def display(self, exam_data):
        # Assume code to display the exam data
        print("Exam Data:")
        print(exam_data)
```

test_module.py

```
from university_package.result import Result
from university_package.college.exam import Exam

def test_function():
    # Access University package
    university_result = Result()
    result_data = university_result.get_data()
    university_result.display(result_data)

    # Access College package
    college_exam = Exam()
    exam_data = college_exam.get_data()
    college_exam.display(exam_data)

# Test the functionality
test_function()
```

10. Create a child class Bus that will inherit all of the variables and methods of the Vehicle class

Answer:-

```
class Vehicle:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

    def drive(self):
        print("The vehicle is driving.")

    def stop(self):
        print("The vehicle has stopped.")



class Bus(Vehicle):
    def __init__(self, brand, color, capacity):
        super().__init__(brand, color)
        self.capacity = capacity

    def open_doors(self):
        print("The bus doors are opened.")

    def close_doors(self):
        print("The bus doors are closed.")



# Create an instance of the Bus class
my_bus = Bus("Mercedes", "Blue", 50)
```

```
# Access the inherited attributes from the Vehicle class
print("Brand:", my_bus.brand)
print("Color:", my_bus.color)

# Call the inherited methods from the Vehicle class
my_bus.drive()
my_bus.stop()

# Call the methods specific to the Bus class
my_bus.open_doors()
my_bus.close_doors()
```

11. Create a Bus class that inherits from the Vehicle class. Give the capacity argument of Bus.seating_capacity() a default value of 50.

Answer:-

```
class Vehicle:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"Make: {self.make}")
        print(f"Model: {self.model}")



class Bus(Vehicle):
    def seating_capacity(self, capacity=50):
        print(f"The seating capacity of this bus is {capacity}.")



# Creating an instance of the Bus class
my_bus = Bus("Volvo", "B7R")


# Accessing the inherited methods from the Vehicle class
my_bus.display_info()


# Calling the seating_capacity method with default value
my_bus.seating_capacity()


# Calling the seating_capacity method with a custom value
my_bus.seating_capacity(60)
```

12. 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.

Answer:-

```
class Vehicle:
    def __init__(self, make, model, seating_capacity):
        self.make = make
        self.model = model
        self.seating_capacity = seating_capacity

    def display_info(self):
        print(f"Make: {self.make}")
        print(f"Model: {self.model}")
        print(f"Seating Capacity: {self.seating_capacity}")

    def fare_charge(self):
        return self.seating_capacity * 100

class Bus(Vehicle):
    def fare_charge(self):
        base_fare = super().fare_charge()
        maintenance_charge = base_fare * 0.1
        total_fare = base_fare + maintenance_charge
        return total_fare

# Creating an instance of the Bus class
my_bus = Bus("Volvo", "B7R", 50)

# Accessing the inherited methods from the Vehicle class
```

```
my_bus.display_info()


# Calculating the fare charge for the bus

fare = my_bus.fare_charge()

print(f"The fare charge for the bus is: {fare}")
```

13.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.

Answer:-

```
class Student:
    def __init__(self, student_name, marks):
        self.student_name = student_name
        self.marks = marks


# Creating an instance of the Student class
student = Student("John Doe", 85)


# Printing the original attribute values
print("Original values:")
print(f"Student Name: {student.student_name}")
print(f"Marks: {student.marks}")


# Modifying the attribute values
student.student_name = "Jane Smith"
student.marks = 92


# Printing the modified attribute values
print("\nModified values:")
print(f"Student Name: {student.student_name}")
print(f"Marks: {student.marks}")
```

14. Write a Python program to match a string that contains only upper and lowercase letters, numbers, and underscores.

Answer:-

```
import re


def match_string(input_string):
    pattern = r'^[a-zA-Z0-9_]+$'
    if re.match(pattern, input_string):
        return True
    else:
        return False


# Test cases
strings = [
    "Hello_World123",
    "hello_world",
    "Hello123",
    "123456",
    "hello world",
    "Hello-World",
    "Hello_World!",
    "HELLO_WORLD"
]


for string in strings:
    if match_string(string):
        print(f"'{string}' matches the pattern.")
    else:
        print(f"'{string}' does not match the pattern.")
```

15. 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

Answer:-

```
import re


def validate_password(password):
    # Check length requirement
    if len(password) < 8 or len(password) > 18:
        return False


    # Check complexity requirements
    pattern = r'^(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%^&*()_\-+=<>?])[a-zA-Z0-9!@#$%^&*()_\-+=<>?]+$'
    if re.match(pattern, password):
        return True
    else:
        return False


# Test cases
passwords = [
    "Abcdefg1!",
    "password123",
    "P@ssw0rd",
    "hello123$",
    "Short!1",
    "Longpassword!1234567890",
    "ComplexPass!word123"
]
```

```
for password in passwords:
    if validate_password(password):
        print(f"'{password}' is a valid password.")
    else:
        print(f"'{password}' is not a valid password.")
```

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

Answer:-

```
import re


def validate_url(url):
    pattern = r'^(https?|ftp)://[^\s/$.?#].[^\s]*$'
    if re.match(pattern, url):
        return True
    else:
        return False


# Test cases
urls = [
    "https://www.example.com",
    "http://www.example.com",
    "ftp://www.example.com",
    "https://example.com",
    "http://example.com",
    "ftp://example.com",
    "https://www.example.com/page?param=value",
    "http://www.example.com/page?param=value",
]


for url in urls:
    if validate_url(url):
        print(f"'{url}' is a valid URL.")
    else:
        print(f"'{url}' is not a valid URL.")
```

17 Write a python program to validate an email address by using regular expression.

Answer:-

```
import re


def validate_email(email):
    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 = [
    "john.doe@example.com",
    "jane_smith@example.co.uk",
    "test123@example",
    "invalid.email@",
    "user@example_com",
    "user@.com",
    "user@example..com",
    "user@-example.com",
    "user@example+domain.com"
]


for email in emails:
    if validate_email(email):
        print(f"'{email}' is a valid email address.")
    else:
        print(f"'{email}' is not a valid email address.")
```

18.Write a python program which consists of - try, except, else, finally blocks.

Answer:-

```
def divide_numbers(num1, num2):
    try:
        result = num1 / num2
        print(f"The division result is: {result}")
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
    else:
        print("Division completed successfully.")
    finally:
        print("Program execution completed.")

# Test cases
divide_numbers(10, 2)
print("--------------------")
divide_numbers(5, 0)
```

19 Write a python program which raises the exception with a message

Answer:-

```
def validate_age(age):
    if age < 0:
        raise ValueError("Age cannot be negative.")
    elif age > 120:
        raise ValueError("Age cannot be greater than 120.")
    else:
        print("Age is valid.")


# Test cases
try:
    validate_age(25)
    validate_age(-10)
    validate_age(150)
except ValueError as e:
    print(f"Error: {str(e)}")
```

20. Write a python program to accept age from user, raise user defined exception if age is below 18 years.

Answer:-

```
class AgeBelow18Error(Exception):
    def __init__(self, message="Age is below 18 years."):
        self.message = message
        super().__init__(self.message)


def validate_age(age):
    if age < 18:
        raise AgeBelow18Error()


try:
    age = int(input("Enter your age: "))
    validate_age(age)
    print("Age is valid.")
except AgeBelow18Error as e:
    print(f"Error: {str(e)}")
except ValueError:
    print("Error: Invalid input. Please enter a valid age.")
```

21. Write a Python multithreading program to print the thread name and corresponding process for each task (assume that there are four tasks).

Answer:-

```
import threading

import os


def task():

    thread_name = threading.current_thread().name

    process_id = os.getpid()

    print(f"Task executed by thread '{thread_name}' in process {process_id}.")


# Create four threads

threads = []

for i in range(4):

    t = threading.Thread(target=task)

    threads.append(t)


# Start the threads

for t in threads:

    t.start()


# Wait for all threads to complete

for t in threads:

    t.join()
```

22 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.

Answer:-

```
import threading


def calculate_square(number):
    result = number ** 2
    print(f"Square of {number} is {result}.")


def calculate_cube(number):
    result = number ** 3
    print(f"Cube of {number} is {result}.")


# Number for calculation
number = 5


# Create two threads
square_thread = threading.Thread(target=calculate_square, args=(number,))
cube_thread = threading.Thread(target=calculate_cube, args=(number,))


# Start the threads
square_thread.start()
cube_thread.start()


# Wait for both threads to complete
square_thread.join()
cube_thread.join()
```

23.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.

Answer:

```
with open("myfile.txt", "r") as file:
    content = file.read()


# Transform the content by writing each word on a separate line
transformed_content = "\n".join(content.split())


# Write the transformed content back to the file
with open("myfile.txt", "w") as file:
    file.write(transformed_content)
```

24. Write a Python program that displays the longest word found in a text file.

Answer:-

```
def find_longest_word(file_path):
    longest_word = ''

    with open(file_path, 'r') as file:
        content = file.read()
        words = content.split()

        for word in words:
            if len(word) > len(longest_word):
                longest_word = word

    return longest_word

# Path to the text file
file_path = 'myfile.txt'

# Find the longest word
longest_word = find_longest_word(file_path)

# Display the longest word
print(f"The longest word in the file is: {longest_word}")
```

25 Write a function in python that allows you to count the frequency of repetition of each word found in a given file.

Answer:-

```
def count_word_frequency(file_path):
    word_frequency = {}

    with open(file_path, 'r') as file:
        content = file.read()
        words = content.split()

        for word in words:
            if word in word_frequency:
                word_frequency[word] += 1
            else:
                word_frequency[word] = 1

    return word_frequency

# Path to the text file
file_path = 'example.txt'

# Count word frequency
frequency = count_word_frequency(file_path)

# Display the word frequency
for word, count in frequency.items():
    print(f"{word}: {count}")
```

26. 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.

Answer:-

```
def extract_lines(file_path, start_line, end_line, output_file):
    with open(file_path, 'r') as file:
        lines = file.readlines()

        # Adjust start and end line indices to account for zero-based indexing
        start_line -= 1
        end_line -= 1

        # Ensure start_line and end_line are within valid range
        start_line = max(0, start_line)
        end_line = min(len(lines) - 1, end_line)

        # Extract the desired lines
        extracted_lines = lines[start_line:end_line+1]

    # Write the extracted lines to the output file
    with open(output_file, 'w') as output:
        output.writelines(extracted_lines)

# Path to the input file
input_file_path = 'input.txt'

# Line range to extract
start_line = 3
end_line = 7
```

```
# Path to the output file
output_file_path = 'extract_content.txt'

# Extract the lines and save them to the output file
extract_lines(input_file_path, start_line, end_line, output_file_path)

print(f"Lines {start_line} to {end_line} extracted and saved to {output_file_path}.")
```

27.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.

Answer:-

```
import pandas as pd


# Create the DataFrame Sales

data = {

   '2018': [110, 205, 177, 189],

   '2019': [130, 165, 175, 190],

   '2020': [115, 206, 157, 179],

   '2021': [118, 198, 183, 169]

}


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


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


# Display the row labels of Sales
```

```
row_labels = sales.index
print("Row Labels:")
print(row_labels)
print()

# Display the column labels of Sales
column_labels = sales.columns
print("Column Labels:")
print(column_labels)
print()

# Display the data types of each column of Sales
data_types = sales.dtypes
print("Data Types:")
print(data_types)
print()

# Display the dimensions, shape, size, and values of Sales
dimensions = sales.ndim
shape = sales.shape
size = sales.size
values = sales.values

print("Dimensions:", dimensions)
print("Shape:", shape)
print("Size:", size)
print("Values:")
print(values)
```

28.Plot the following data on a line chart and customize the chart according to the below-given 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

Answer:-

```
import matplotlib.pyplot as plt

# Data
months = ['January', 'February', 'March', 'April', 'May']
sales = [510, 350, 475, 580, 600]

# Customize the line chart
plt.plot(months, sales, color='blue', linestyle='--', marker='D')

# Set chart title and axis labels
plt.title("The Monthly Sales Report")
plt.xlabel("Month")
plt.ylabel("Sales")

# Display legends
plt.legend(["Sales"])
```

```
# Display the line chart

plt.show()
```

29. 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

2.1 . Use different widths

2.2. Use different colors to represent different years score

2.3. Display appropriate titles for axis and chart

2.4. Show legends

2.5. Create a bar chart to display data of Steve Smith, Kane Williamson

& Jos Butler. Customize Chart as per your wish.

2.6. Display data of all players for the specific year.

Answer:-

```
import matplotlib.pyplot as plt


# Data

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

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

virat_scores = [2501, 1855, 2203, 1223]

steve_scores = [2340, 2250, 2003, 1153]

babar_scores = [1750, 2147, 1896, 1008]

rohit_scores = [1463, 1985, 1854, 1638]

kane_scores = [1256, 1785, 1874, 1974]
```

```
jos_scores = [1125, 1853, 1769, 1436]

# Set custom widths for the bars
bar_width = 0.15

# Create a bar chart for Virat Kohli and Rohit Sharma
plt.bar(years, virat_scores, width=bar_width, color='blue', label='Virat Kohli')
plt.bar(years, rohit_scores, width=bar_width, color='red', label='Rohit Sharma')

# Set chart title and axis labels
plt.title("Batsmen Performance Over Years")
plt.xlabel("Year")
plt.ylabel("Runs Scored")

# Display legends
plt.legend()

# Display the bar chart for Steve Smith, Kane Williamson, and Jos Butler
x_pos = [i + bar_width for i in range(len(years))]

plt.bar(x_pos, steve_scores, width=bar_width, color='green', label='Steve Smith')
plt.bar(x_pos, kane_scores, width=bar_width, color='orange', label='Kane Williamson')
plt.bar(x_pos, jos_scores, width=bar_width, color='purple', label='Jos Butler')

# Update x-axis ticks and labels
plt.xticks([i + bar_width for i in range(len(years))], years)

# Display legends
plt.legend()
```

```
# Show the chart

plt.show()


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

specific_year = '2019'


# Get the index of the specific year

year_index = years.index(specific_year)


# Get the scores of all players for the specific year

specific_year_scores = [virat_scores[year_index], steve_scores[year_index],
babar_scores[year_index], rohit_scores[year_index], kane_scores[year_index], jos_scores[year_index]]


# Create a bar chart for the specific year

plt.bar(batsmen, specific_year_scores, color='maroon')


# Set chart title and axis labels

plt.title("Runs Scored in " + specific_year)

plt.xlabel("Batsman")

plt.ylabel("Runs Scored")


# Show the chart

plt.show()
```

30. Write a program 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

Answer:-

```
import numpy as np


# Create a 3x3 NumPy array with user input
array = np.zeros((3, 3))


for i in range(3):
    for j in range(3):
        num = int(input(f"Enter element at position ({i}, {j}): "))
        array[i, j] = num


# Calculate the sum of all elements in the array
array_sum = np.sum(array)


# Print the array and the sum
print("Array:")
print(array)
print("Sum of all elements:", array_sum)
```

31. Write a program to perform basic arithmetic operations on 1D and 2D array.

Answer:-

```
import numpy as np

# 1D array
arr1 = np.array([1, 2, 3, 4, 5])

# 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Addition
addition_1d = arr1 + 10
addition_2d = arr2 + 5

# Subtraction
subtraction_1d = arr1 - 2
subtraction_2d = arr2 - 3

# Multiplication
multiplication_1d = arr1 * 3
multiplication_2d = arr2 * 2

# Division
division_1d = arr1 / 2
division_2d = arr2 / 4

# Print the results
print("1D Array:")
print("Original Array:", arr1)
print("Addition:", addition_1d)

print("Subtraction:", subtraction_1d)

print("Multiplication:", multiplication_1d)

print("Division:", division_1d)


print("\n2D Array:")

print("Original Array:")

print(arr2)

print("Addition:")

print(addition_2d)

print("Subtraction:")

print(subtraction_2d)

print("Multiplication:")

print(multiplication_2d)

print("Division:")

print(division_2d)
```

32. 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.

Answer:-

```
from pymongo import MongoClient


client = MongoClient("mongodb://localhost:27017/")
database = client["student_db"]
collection = database["students"]




def add_student():
    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!")



def display_students():
    students = collection.find()
    for student in students:
        print(f"Student ID: {student['student_id']}")
        print(f"Student Name: {student['student_name']}")
        print(f"Course: {student['course']}")
        print()


def update_student():
    student_id = input("Enter Student ID to update: ")
    new_student_name = input("Enter New Student Name: ")
    new_course = input("Enter New Course: ")

    update_query = {
        "student_id": student_id
    }

    new_values = {
        "$set": {
            "student_name": new_student_name,
            "course": new_course
        }
    }

    collection.update_one(update_query, new_values)
    print("Student updated successfully!")


def delete_student():
    student_id = input("Enter Student ID to delete: ")
    delete_query = {
        "student_id": student_id
    }

    collection.delete_one(delete_query)
    print("Student deleted successfully!")

while True:
    print("Student Database Menu:")
    print("1. Add Student")
    print("2. Display Students")
    print("3. Update Student")
    print("4. Delete Student")
    print("5. Exit")

    choice = input("Enter your choice (1-5): ")

    if choice == "1":
        add_student()
    elif choice == "2":
        display_students()
    elif choice == "3":
        update_student()
    elif choice == "4":
        delete_student()
    elif choice == "5":
        break
    else:
            print("Invalid choice! Please try again.")

        print()

    if __name__ == "__main__":
        print("Welcome to Student Database!")
        print("-----------------------------")
        print()

        while True:
            username = input("Enter MongoDB username: ")
            password = input("Enter MongoDB password: ")

            try:
                client =
MongoClient(f"mongodb+srv://{username}:{password}@cluster0.mongodb.net/")
                database = client["student_db"]
                collection = database["students"]
                break
            except:
                print("Connection failed! Please try again.")

        print("Connection successful!")
        print()

        # Run the main menu loop
        while True:
            print("Student Database Menu:")
            print("1. Add Student")
print("2. Display Students")

print("3. Update Student")

print("4. Delete Student")

print("5. Exit")


choice = input("Enter your choice (1-5): ")


if choice == "1":

    add_student()

elif choice == "2":

    display_students()

elif choice == "3":

    update_student()

elif choice == "4":

    delete_student()

elif choice == "5":

    break

else:

    print("Invalid choice! Please try again.")


print()
```

33. Demonstrate step by step MongoDB connection in Python

Answer:-

```
# Import the necessary module
from pymongo import MongoClient

# Establish a connection with the MongoDB server
client = MongoClient("mongodb://localhost:27017/")
# Replace "localhost" with the IP address or hostname of the MongoDB server
# Replace "27017" with the port number where MongoDB is running

# Access a specific database
database = client["mydatabase"]
# Replace "mydatabase" with the name of the database you want to access

# Access a specific collection within the database
collection = database["mycollection"]
# Replace "mycollection" with the name of the collection you want to access

# Perform database operations
# Example: Insert a document into the collection
document = {
   "name": "John Doe",
   "age": 30,
   "city": "New York"
}
collection.insert_one(document)
# This will insert the document into the specified collection

# Example: Query documents from the collection
```

```
query = {"name": "John Doe"}

results = collection.find(query)

for result in results:

    print(result)

# This will find all documents in the collection that match the given query and print them


# Example: Update a document in the collection

filter = {"name": "John Doe"}

update = {"$set": {"age": 35}}

collection.update_one(filter, update)

# This will update the first document in the collection that matches the given filter


# Example: Delete a document from the collection

filter = {"name": "John Doe"}

collection.delete_one(filter)

# This will delete the first document in the collection that matches the given filter


# Close the MongoDB connection

client.close()
```

34.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

Answer:-

```
from pymongo import MongoClient


# Establish a connection with the MongoDB server
client = MongoClient("mongodb://localhost:27017/")


# Access the book database
database = client["bookdb"]


# Access the books collection
collection = database["books"]


# Function to add a book to the database
def add_book():
    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 books in the database
def display_books():
    books = collection.find()
    for book in books:
        print(book)

# Function to search for a book by its name
def search_book():
    book_name = input("Enter Book Name to search: ")
    query = {"Book_name": book_name}
    book = collection.find_one(query)
    if book:
        print(book)
    else:
        print("Book not found!")

# Function to sort books by book name
def sort_books():
    books = collection.find().sort("Book_name")
    for book in books:
        print(book)

# Menu-driven program loop
while True:
    print("\nBOOK DATABASE MENU:")
    print("1. Add Book")
    print("2. Display Books")
    print("3. Search Book")
    print("4. Sort Books")
    print("5. Exit")

    choice = input("Enter your choice (1-5): ")

    if choice == "1":
        add_book()
    elif choice == "2":
        display_books()
    elif choice == "3":
        search_book()
    elif choice == "4":
        sort_books()
    elif choice == "5":
        break
    else:
        print("Invalid choice! Please try again.")

# Close the MongoDB connection
client.close()
```