Practical Set – 1

1. A program that converts temperatures from Fahrenheit to Celsius and vice versa.
2. A program that calculates the area and perimeter of a rectangle.
3. A program that generates a random password of a specified length.
4. A program that calculates the average of a list of numbers.
5. A program that checks if a given year is a leap year.
6. A program that calculates the factorial of a number.
7. A program that checks if a given string is a palindrome.
8. A program that sorts a list of numbers in ascending or descending order.
9. A program that generates a multiplication table for a given number.
10. A program that converts a given number from one base to another.

Code:

```
def convTemp():
    print("1. Fahrenheit to Celsius")
    print("2. Celsius to Fahrenheit")
    option = int(input("Select the conversion: "))
 if option == 1:
    far = float(input("Enter temperature in Fahrenheit: "))
    cel = (far - 32) * 5 / 9
    print(f"Fahrenheit {far} = Celsius {cel:.2f}")
elif option == 2:
 cel = float(input("Enter temperature in Celsius: "))
far = (cel * 9 / 5) + 32
print(f"Celsius {cel} = Fahrenheit {far:.2f}")
else:
print("Invalid option, please choose 1 or 2.")
def rectangleProperties():
    length = float(input("Enter the length of the rectangle: "))
    width = float(input("Enter the width of the rectangle: "))
    area = length * width
    perimeter = 2 * (length + width)
    print(f"Area = {area}, Perimeter = {perimeter}")
def generatePassword():
    length = int(input("Enter the desired password length: "))
    characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()"
    password = "".join(random.choice(characters) for _ in range(length))
```

```
        print(f"Generated password: {password}")
def avgList():
        myList = []

        while True:
            num = int(input("Enter a number: "))
            myList.append(num)

            ch = input("Do you want to add more numbers (yes/no)?
    ").lower()
            if ch == "no":
                break

        print("List of numbers:", myList)
        total_sum = sum(myList)
        print(f"Sum = {total_sum}, Average = {total_sum /
    len(myList):.2f}")

    def sortList():
        myList = []
        while True:
            num = int(input("Enter a number: "))
            myList.append(num)

            ch = input("Do you want to add more numbers (yes/no)?
    ").lower()
            if ch == "no":
                break

        print("Original list:", myList)
        order = input("Sort in ascending or descending order? (asc/desc):
    ").lower()
        if order == "asc":
            myList.sort()
        elif order == "desc":
            myList.sort(reverse=True)
        else:
            print("Invalid order choice. Defaulting to ascending.")
            myList.sort()

        print("Sorted list:", myList)
def multiTable():
        num = int(input("Enter the number for the multiplication table: "))
        for i in range(1, 11):
            print(f"{num} x {i} = {num * i}")
```

```
def checkLeapYear():
    year = int(input("Enter a year: "))
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        print(f"{year} is a leap year.")
    else:
        print(f"{year} is not a leap year.")

def calculateFactorial():
    num = int(input("Enter a number: "))
    factorial = 1
    for i in range(1, num + 1):
        factorial *= i
    print(f"Factorial of {num} is {factorial}")

def checkPalindrome():
    string = input("Enter a string: ")
    if string == string[::-1]:
        print(f"'{string}' is a palindrome.")
    else:
        print(f"'{string}' is not a palindrome.")

def baseConversion():
    num = int(input("Enter a number: "))
    from_base = int(input("Enter the base of the number (2 for binary,
10 for decimal, etc.): "))
    to_base = int(input("Enter the base you want to convert to (2 for
binary, 10 for decimal, etc.): "))

    if from_base == 10:
        if to_base == 2:
            print(f"{num} in decimal is {bin(num)[2:]} in binary.")
        elif to_base == 16:
            print(f"{num} in decimal is {hex(num)[2:]} in
hexadecimal.")
        else:
            print("Unsupported conversion.")
    else:
        print("Unsupported base conversion.")
```

```
def main():
    while True:
        print("\nMenu:")
        print("0. Exit")
        print("1. Convert temperature")
        print("2. Calculate area and perimeter of a rectangle")
        print("3. Generate a random password")
        print("4. Calculate average of numbers in a list")
        print("5. Check if a year is a leap year")
        print("6. Calculate factorial")
        print("7. Check if a string is a palindrome")
        print("8. Sort a list")
        print("9. Print multiplication table")
        print("10. Convert number between bases")

        option = int(input("Select your option: "))

        if option == 0:
            print("Thank you! Goodbye!")
            break
        elif option == 1:
            convTemp()
        elif option == 2:
            rectangleProperties()
        elif option == 3:
            generatePassword()
        elif option == 4:
            avgList()
        elif option == 5:
            checkLeapYear()
        elif option == 6:
            calculateFactorial()
        elif option == 7:
            checkPalindrome()
        elif option == 8:
            sortList()
        elif option == 9:
            multiTable()
        elif option == 10:
            baseConversion()
        else:
            print("Invalid option. Please try again.")
```

```
if __name__ == "__main__":
    main()
```

Output:

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 1
Enter temperature in Fahrenheit: 98
Fahrenheit 98.0 = Celsius 36.67
Enter temperature in Celsius: 36.67
Celsius 36.67 = Fahrenheit 98.01
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 2
Enter the length of the rectangle: 2
Enter the width of the rectangle: 3
Area = 6.0, Perimeter = 10.0
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 3
Enter the desired password length: 9
Generated password: Tj8&lT!E%
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 4
Enter a number: 3
Do you want to add more numbers (yes/no)? yes
Enter a number: 7
Do you want to add more numbers (yes/no)? yes
Enter a number: 18
Do you want to add more numbers (yes/no)? no
List of numbers: [3, 7, 18]
Sum = 28, Average = 9.33
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 5
Enter a year: 2025
2025 is not a leap year.
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 6
Enter a number: 7
Factorial of 7 is 5040
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 7
Enter a string: malayalam
'malayalam' is a palindrome.
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 8
Enter a number: 1
Do you want to add more numbers (yes/no)? yes
Enter a number: 20
Do you want to add more numbers (yes/no)? yes
Enter a number: 46
Do you want to add more numbers (yes/no)? yes
Enter a number: 82
Do you want to add more numbers (yes/no)? no
Original list: [1, 20, 46, 82]
Sort in ascending or descending order? (asc/desc): asc
Sorted list: [1, 20, 46, 82]

Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 8
Enter a number: 4
Do you want to add more numbers (yes/no)? yes
Enter a number: 19
Do you want to add more numbers (yes/no)? yes
Enter a number: 1
Do you want to add more numbers (yes/no)? no
Original list: [4, 19, 1]
Sort in ascending or descending order? (asc/desc): desc
Sorted list: [19, 4, 1]
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 9
Enter the number for the multiplication table: 23
23 x 1 = 23
23 x 2 = 46
23 x 3 = 69
23 x 4 = 92
23 x 5 = 115
23 x 6 = 138
23 x 7 = 161
23 x 8 = 184
23 x 9 = 207
23 x 10 = 230
```

```
Menu:
0. Exit
1. Convert temperature
2. Calculate area and perimeter of a rectangle
3. Generate a random password
4. Calculate average of numbers in a list
5. Check if a year is a leap year
6. Calculate factorial
7. Check if a string is a palindrome
8. Sort a list
9. Print multiplication table
10. Convert number between bases
Select your option: 10
Enter a number: 10
Enter the base of the number (2 for binary, 10 for decimal, etc.): 10
Enter the base you want to convert to (2 for binary, 10 for decimal, etc.): 2
10 in decimal is 1010 in binary.
```

Practical Set – 2

Aim 1: A program that models a bank account, with classes for the account, the customer, and the bank.

Code:

```
import random

class Customer:
    def __init__(self, name, address, contact_number):
        self.name = name
        self.address = address
        self.contact_number = contact_number
        self.accounts = []

    def create_account(self, account_type, initial_balance):
        account_number = Bank.generate_account_number()
        account = BankAccount(account_type, initial_balance, self, account_number)
        self.accounts.append(account)
        return account

    def display_customer_info(self):
        print(f"Customer Name: {self.name}")
        print(f"Address: {self.address}")
        print(f"Contact Number: {self.contact_number}")
        print("Accounts:")
        for account in self.accounts:
            print(f" - {account}")

class BankAccount:
    def __init__(self, account_type, balance, owner, account_number):
        self.account_type = account_type
        self.balance = balance
        self.owner = owner
        self.account_number = account_number

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited INR {amount}. New balance: INR {self.balance}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew INR {amount}. New balance: INR {self.balance}")
        else:
            print("Insufficient funds!")

    def __str__(self):
        return f"{self.account_type} Account - Account Number: {self.account_number}, Balance: INR
{self.balance}"

class Bank:
    def __init__(self, name):
        self.name = name
        self.customers = []
```

```
    def add_customer(self, customer):
        self.customers.append(customer)

    @staticmethod
    def generate_account_number():
        return ''.join(random.choice('0123456789') for _ in range(8))

    def display_bank_info(self):
        print(f"Bank Name: {self.name}")
        print("Customers:")
        for customer in self.customers:
            customer.display_customer_info()
            print()

    def find_account_by_number(self, account_number):
        for customer in self.customers:
            for account in customer.accounts:
                if account.account_number == account_number:
                    return account
        return None

# Example usage
if __name__ == "__main__":
    my_bank = Bank("My Bank")
    customer_list = []

    while True:
        try:
            print("\n1. New Customer 2. Existing Customer 3. Find Customer Info 4. Exit")
            choice = int(input("Enter your option: "))

            if choice == 1:
                print("\nCustomer Registration:")
                name = input("Enter Customer Name: ")
                address = input("Enter Customer Address: ")
                contact_number = input("Enter Customer Contact Number: ")
                customer_obj = Customer(name, address, contact_number)
                customer_list.append(customer_obj)
                my_bank.add_customer(customer_obj)

                while True:
                    acc_type = int(input("Enter 1. To create Savings account 2. To create Current account 3. Exit\n"))
                    if acc_type == 1:
                        new_account = customer_obj.create_account("Savings", 1000)
                        print(f"Savings account created with account number: {new_account.account_number}\n")
                        break
                    elif acc_type == 2:
                        new_account = customer_obj.create_account("Current", 1000)
                        print(f"Current account created with account number: {new_account.account_number}\n")
                        break
                    elif acc_type == 3:
                        break
                    else:
                        print("Invalid option... Try again")

            elif choice == 2:
                account_number_input = input("Enter your account number: ")
                account_to_transact = my_bank.find_account_by_number(account_number_input)

                if account_to_transact:
```

```
            print(f"\nWelcome, {account_to_transact.owner.name}!")
            print(account_to_transact)

            while True:
                print("\n1. Deposit\n2. Withdraw\n3. Check Balance\n4. Exit")
                option = int(input("Enter your option: "))

                if option == 1:
                    deposit_amount = int(input("\nEnter the amount to deposit: INR "))
                    account_to_transact.deposit(deposit_amount)
                elif option == 2:
                    withdrawal_amount = int(input("\nEnter the amount to withdraw: INR "))
                    account_to_transact.withdraw(withdrawal_amount)
                elif option == 3:
                    print("\nUpdated Account Information:")
                    print(account_to_transact)
                elif option == 4:
                    break
                else:
                    print("Invalid option")

        else:
            print("Account not found.")

    elif choice == 3:
        my_bank.display_bank_info()

    elif choice == 4:
        break

    else:
        print("Invalid option... Try again.")

except ValueError:
    print("Invalid input. Please enter a valid option.")
```

OUTPUT:

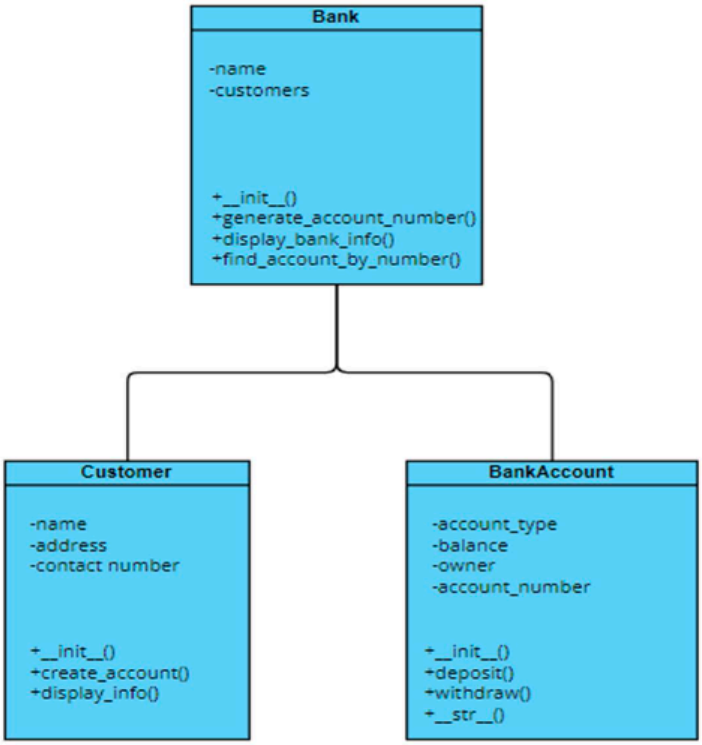
```
1. New Customer 2. Existing Customer 3. Find Customer Info 4. Exit
Enter your option: 1

Customer Registration:
Enter Customer Name: Meet
Enter Customer Address: Vadodara
Enter Customer Contact Number: 9876543210
Enter 1. To create Savings account 2. To create Current account 3. Exit
1
Savings account created with account number: 98161704
```

```
1. New Customer 2. Existing Customer 3. Find Customer Info 4. Exit
Enter your option: 4
```

```
1. New Customer 2. Existing Customer 3. Find Customer Info 4. Exit
Enter your option: 2
Enter your account number: 98161704

Welcome, Meet!
Savings Account - Account Number: 98161704, Balance: INR 1000

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your option: 1

Enter the amount to deposit: INR 10000000
Deposited INR 10000000. New balance: INR 10001000

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your option: 3

Updated Account Information:
Savings Account - Account Number: 98161704, Balance: INR 10001000

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your option: 2

Enter the amount to withdraw: INR 123456
Withdrew INR 123456. New balance: INR 9877544

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your option: 4
```

```
1. New Customer 2. Existing Customer 3. Find Customer Info 4. Exit
Enter your option: 3
Bank Name: My Bank
Customers:
Customer Name: Meet
Address: Vadodara
Contact Number: 9876543210
Accounts:
 - Savings Account - Account Number: 98161704, Balance: INR 9877544
```

Class Diagram:



Practical Set – 2

Aim 2: A program that simulates a school management system, with classes for the students, the teachers, and the courses.

Code:

```
class Student:
        def __init__(self, student_id, name, grade):
        self.student_id = student_id
        self.name = name
        self.grade = grade

      def display_info(self):
      print(f"\nStudent ID: {self.student_id}, Name: {self.name}, Grade: {self.grade}")

  class Teacher:
        def __init__(self, teacher_id, name, subject):
        self.teacher_id = teacher_id
        self.name = name
        self.subject = subject

      def display_info(self):
      print(f"\nTeacher ID: {self.teacher_id}, Name: {self.name}, Subject: {self.subject}")

  class Course:
      def __init__(self, course_code, course_name, teacher, students):
      self.course_code = course_code
      self.course_name = course_name
      self.teacher = teacher
      self.students = students

      def display_info(self):
      print(f"\nCourse Code: {self.course_code}, Course Name: {self.course_name}")
      print("\nTeacher:")
      self.teacher.display_info()
      print("\nStudents:")
      for student in self.students:
      student.display_info()
      def main():
      students = []
      teachers = []
      courses = []

      while True:
          print("\n1. Student Registration/Details\n2. Teacher Registration/Details\n3. Course Registration/Details\n4.
      Exit")
       try:
          choice = int(input("\nEnter your choice: "))

 if choice == 1:
    num_students = int(input("\nEnter the number of students: "))
    for i in range(num_students):
       student_id = input(f"\nEnter student {i + 1} ID: ")
       name = input(f"Enter student {i + 1} name: ")
       grade = input(f"Enter student {i + 1} grade: ")
```

```
        students.append(Student(student_id, name, grade))
    print("\nRegistration successful.")

elif choice == 2:
    num_teachers = int(input("\nEnter the number of teachers: "))
    for i in range(num_teachers):
        teacher_id = input(f"\nEnter teacher {i + 1} ID: ")
        name = input(f"Enter teacher {i + 1} name: ")
        subject = input(f"Enter teacher {i + 1} subject: ")
        teachers.append(Teacher(teacher_id, name, subject))
    print("\nRegistration successful.")

elif choice == 3:
    if not teachers:
        print("\nNo teachers available. Please register teachers first.")
        continue
    if not students:
        print("\nNo students available. Please register students first.")
        continue

    num_courses = int(input("\nEnter the number of courses: "))
    for i in range(num_courses):
        course_code = input(f"\nEnter course {i + 1} code: ")
        course_name = input(f"Enter course {i + 1} name: ")

        print("\nAvailable Teachers:")
        for idx, teacher in enumerate(teachers):
            print(f"{idx}. {teacher.name} - {teacher.subject}")

        try:
            teacher_index = int(input("\nEnter the index of the teacher for this course: "))
            if teacher_index < 0 or teacher_index >= len(teachers):
                print("\nInvalid teacher index.")
                continue
            teacher = teachers[teacher_index]
        except ValueError:
            print("\nInvalid input. Please enter a valid number.")
            continue

        print("\nAvailable Students:")
        for idx, student in enumerate(students):
            print(f"{idx}. {student.name} - {student.grade}")

        student_indices = input("\nEnter the indices of students for this course (comma-separated): ")
        try:
            students_for_course = [students[int(index)] for index in student_indices.split(",") if index.strip().isdigit()]
            if not students_for_course:
                print("\nNo valid students selected. Please try again.")
                continue
        except IndexError:
            print("\nInvalid student index. Please try again.")
            continue

        courses.append(Course(course_code, course_name, teacher, students_for_course))
        print("\nRegistration successful.")

elif choice == 4:
    print("\nExiting program...")
    break
```

```
        else:
            print("\nInvalid input. Please enter a valid option.")

    except ValueError:
        print("\nInvalid input. Please enter a valid number.")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
1. Student Registration/Details
2. Teacher Registration/Details
3. Course Registration/Details
4. Exit

Enter your choice: 1

Enter the number of students: 1

Enter student 1 ID: 1920364758
Enter student 1 name: MEET
Enter student 1 grade: 2nd

Registration successful.

1. Student Registration/Details
2. Teacher Registration/Details
3. Course Registration/Details
4. Exit

Enter your choice: 2

Enter the number of teachers: 1

Enter teacher 1 ID: 1425369708
Enter teacher 1 name: AMIT
Enter teacher 1 subject: PPFSD

Registration successful.
```

```
1. Student Registration/Details
2. Teacher Registration/Details
3. Course Registration/Details
4. Exit

Enter your choice: 3

Enter the number of courses: 1

Enter course 1 code: 1627384950
Enter course 1 name: PPFSD

Available Teachers:
0. AMIT - PPFSD

Enter the index of the teacher for this course: 0

Available Students:
0. MEET - 2nd

Enter the indices of students for this course (comma-sepa

Registration successful.

1. Student Registration/Details
2. Teacher Registration/Details
3. Course Registration/Details
4. Exit

Enter your choice: 4

Exiting program...
```

Practical Set – 2

Aim 3: A program that reads a text file and counts the number of words in it.

Code:

```
def count(path):
    try:
      with open(path, 'r') as file:
          file_content = file.read()
          return f"data = {file_content.split()}\nlength of the words: {len(file_content.split())}"
    except FileNotFoundError:
          return "Please provide a valid file path."

path = "example.txt"
print(count(path))
```

```
data = ['Hey!', 'MEET', 'HERE!']
length of the words: 3
```

Practical Set – 2

Aim 4: A program that reads a CSV file and calculates the average of the values in a specified column.

Code:

```
import csv

def calculate_average(csv_file, column_name):
    try:
        with open(csv_file, 'r') as file:
            reader = csv.DictReader(file)
            if not reader.fieldnames:
                print(f"Error: The CSV file '{csv_file}' is empty or improperly formatted.")
                return None
            if column_name not in reader.fieldnames:
                print(f"Column '{column_name}' not found in the CSV file.")
                return None

            total = 0
            count = 0

            # Iterate through rows to calculate the sum and count valid values
            for row in reader:
                try:
                    value = float(row[column_name])
                    total += value
                    count += 1
                except ValueError:
                    print(f"Skipping row {reader.line_num}: Invalid value in column '{column_name}'.")
            if count == 0:
                print(f"No valid values found in column '{column_name}'.")
                return None

            average = total / count
            return average

    except FileNotFoundError:
        print(f"Error: File '{csv_file}' not found.")
        return None
    except Exception as e:
        print(f"Unexpected error: {e}")
        return None
csv_file_path = 'file.csv'
column_to_calculate = 'ENGLISH'

result = calculate_average(csv_file_path, column_to_calculate)
if result is not None:
    print(f"The average value in column '{column_to_calculate}' is: {result:.2f}")
```

```
/avg.  of values in specified column.py"
Skipping row 4: Invalid value in column 'ENGLISH'.
The average value in column 'ENGLISH' is: 88.50
```

Practical Set – 2

Aim 5: A program that reads an Excel file and prints the data in a tabular format.

Code:

```
import pandas as pd

file_path = "/Users/meetlimbachiya/Desktop/CODE/LAB /SET 2/delimited.xlsx"

try:
    df = pd.read_excel(file_path, engine="openpyxl")
    print(df.to_string(index=False))
except FileNotFoundError:
    print(f"Error: File '{file_path}' not found.")
except Exception as e:
    print(f"Unexpected error: {e}")
```

```
Sr. NO. Name     ERP  MATHS  CN  OS  PPFSD
      1 Amit 1234567     23  25  22     22
      2  mit 1234568     24  21  23     24
      3 meet 1234569     22  22  21     22
      4  kit 1234570     21  24  24     24
      5  kat 1234571     25  23  25     23
```

Practical Set – 3

Aim 1: A program that creates a simple web server and serves a static HTML page.

Code:

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Static HTML Page</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

FLASK:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)
```

Hello World!

SET-3

PRACTICAL-1

- **AIM : A program that creates a simple web server and serves a static HTMLpage.**

- CODE:

  - index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Static HTML Page</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

- app.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route("/")
def home():
    return render_template("index.html")
if___name___== "__main__":
        app.run(debug=True)
```

Actual Output:

PRACTICAL- 2

- **AIM: A program that creates a web application that allows users to registerand login.**

- CODE:

  - Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Static HTML Page</title>
  </head>
  <style>
    @import
   url("https://fonts.googleapis.com/css2?family=Poppins:wght@500&display=swap");
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
    body {
      height: 100vh;
      width: 100%;
      display: flex;
      justify-content: center;
      align-items: center;
      flex-direction: column;
      background: #ff5a5f;
    }
    h1 {
      font-family: "Poppins", sans-serif;
      color: #fff;
      margin: 30px 50px;
      font-size: 3rem;
    }
    input {
      padding: 10px 20px;
      border: 3px solid #fff;
      border-radius: 10px;
      background: rgb(16, 208, 16);
      font-size: 1.5rem;
      color: white;
```

```
            font-family: "Poppins", sans-serif;
            font-weight: 300;
            transition: .3s;
            &:hover{
            background: #fff;
            color: #000;
            cursor: pointer;
              }
          }
        </style>
        <body>
          <h1>Hello, this is a static HTML page served by Flask!</h1>
          <form action="{{ url_for('register') }}">
            <input type="submit" value="Register" />
          </form>
        </body>
    </html>
```

- login.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>User Login</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        height: 100vh;
        width: 100%;
        display: flex;
        align-items: center;
        justify-content: center;
        flex-direction: column;
        background: rgb(9, 9, 121);
        background: linear-gradient(
        30deg,
          rgba(9, 9, 121, 1) 0%,
          rgba(2, 0, 36, 1) 29%,
          rgba(0, 212, 255, 1) 100%
        );
      }
      .container {
```

```
    display: flex;
  align-items: center;

  justify-content: space-evenly;
  flex-direction: column;
  width: 600px;
  border-radius: 20px;
  height: 500px;
  background: #ffffff5a;
backdrop-filter: blur(20px);
  & h1 {
    font-family: Arial, Helvetica, sans-serif;
    color: #fff;
    margin: 30px 0;
  }
  & li {
    list-style: none;
  }
  & form {
    & label {
      color: white;
      font-family: Arial, Helvetica, sans-serif;
      font-size: 1.4rem;
      margin: 10px 20px;
    }
    & .log_button {
      color: #fff;
      background: red;
      border: none;
      outline: none;
      padding: 5px 10px;
      border-radius: 10px;
      font-size: 1.2rem;
      transition: 0.3s;
      transform: translateX(130px);
      &:hover {
        background: #fff;
        color: #000;
        cursor: pointer;
      }
  }
  &  .password{
      padding: 10px 20px;
      border-radius: 20px;
      outline: none;
      border: none;
  }
  &  .username{
      padding: 10px 20px;
      border-radius: 20px;
```

```
            outline: none;
            border: none;
          }
          & input {

            margin: 10px 20px;
          }
          }
        }
        .error {
          color: red;
        }
        .success {
          color: green;
        }
        .default {
          color: black;
        }
      </style>
    </head>
    <body>
      <div class="container">
        <h1>User Login</h1>
        {% with messages = get_flashed_messages() %} {% if messages %}
        <ul>
          {% for message in messages %}
          <li
            class="{% if 'error' in message %}error{% elif 'success' in message %}success{% else
      %}default{% endif
    %}"
          >
            {{ message }}
          </li>
          {% endfor %}
        </ul>
        {% endif %} {% endwith %}
        <form method="post" action="{{ url_for('login') }}">
          <label for="username" class="username_label">Username:</label>
          <input type="text" name="username" class="username" required />
          <br />
          <label for="password" class="password_label">Password:</label>
          <input type="password" name="password" class="password" required />
          <br />
          <input type="submit" class="log_button" value="Log in" />
        </form>
        <p>
          Don't have an account?
          <a href="{{ url_for('register') }}">Register here</a>.
        </p>
      </div>
    </body>
```

```
</html>
```

- register.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>User Registration</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        height: 100vh;
        width: 100%;
        display: flex;
        align-items: center;
        justify-content: center;
        flex-direction: column;
        background: rgb(9, 9, 121);
        background: linear-gradient(
        30deg,
          rgba(9, 9, 121, 1) 0%,
          rgba(2, 0, 36, 1) 29%,
          rgba(0, 212, 255, 1) 100%
        );
      }
      .container {
        display: flex;
        align-items: center;
        justify-content: space-evenly;
        flex-direction: column;
        width: 600px;
        border-radius: 20px;
        height: 500px;
        background: #ffffff5a;
        backdrop-filter: blur(20px);
        & h1 {
          font-family: Arial, Helvetica, sans-serif;
          color: #fff;
          margin: 30px 0;
        }
        & li {
          list-style: none;
        }
```

```
& form {

  & label {
        color: white;


        font-family: Arial, Helvetica, sans-
    serif; font-size: 1.4rem;
    margin: 10px 20px;
  }
  & .register_button {
    color: #fff;
    background: red;
    border: none;
    outline: none;
    padding: 5px 10px;
    border-radius: 10px;
    font-size: 1.2rem;
    transition: 0.3s;
    transform: translateX(130px);
    &:hover {
      background: #fff;
      color: #000;
      cursor: pointer;
    }
  }
  & .password {
    padding: 10px 20px;
    border-radius: 20px;
    outline: none;
    border: none;
  }
  & .username {
    padding: 10px 20px;
    border-radius: 20px;
    outline: none;
    border: none;
  }
  & input {
    margin: 10px 20px;
  }
 }
}
.error {
 color: red;
}
.success {
 color: green;
}
```

```
      .default {
        color: black;
      }
    </style>
  </head>
  <body>
    <div class="container">

    <h1>User Registration</h1>

      {% with messages = get_flashed_messages() %}  {% if messages %}
      <ul>
        {% for message in messages %}
        <li
          class="{% if 'error' in message %}error{% elif 'success' in message %}success{% else
    %}default{% endif
%}"
        >
          {{ message }}
        </li>
        {% endfor %}
      </ul>
      {% endif %}  {% endwith %}
      <form method="post" action="{{ url_for('register') }}">
        <label for="username" class="username_label">Username:</label>
        <input type="text" name="username" class="username" required />
        <br />
        <label for="password" class="password_label">Password:</label>
        <input type="password" name="password" class="password" required />
        <br />
        <input type="submit" class="register_button" value="Register" />
      </form>
      <p>
        Already have an account?
        <a href="{{ url_for('login') }}">Log in here</a>.
      </p>
    </div>
  </body>
</html>
```

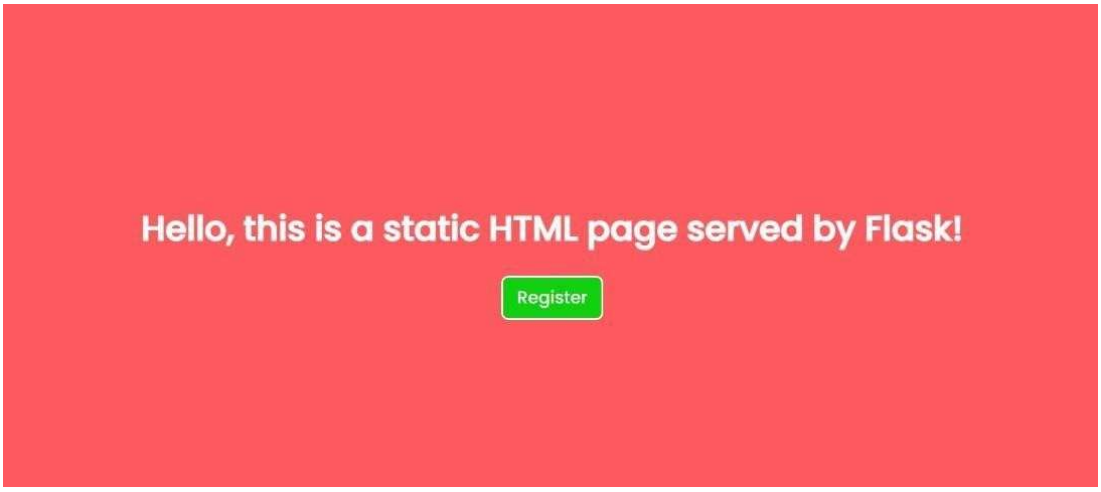
- app.py

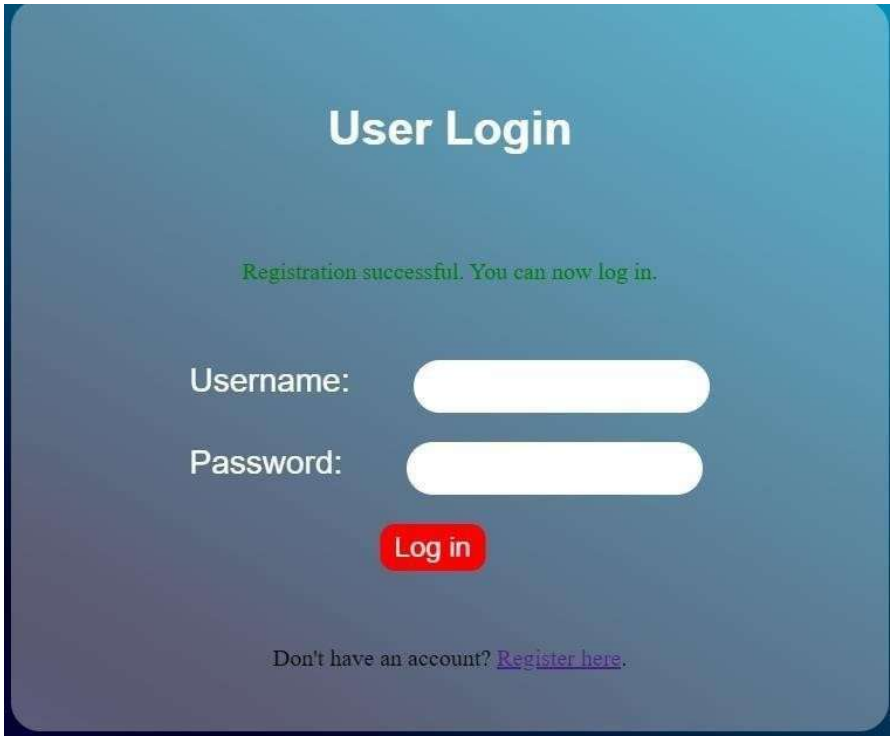
```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
import secrets

app = Flask(__name__)
```

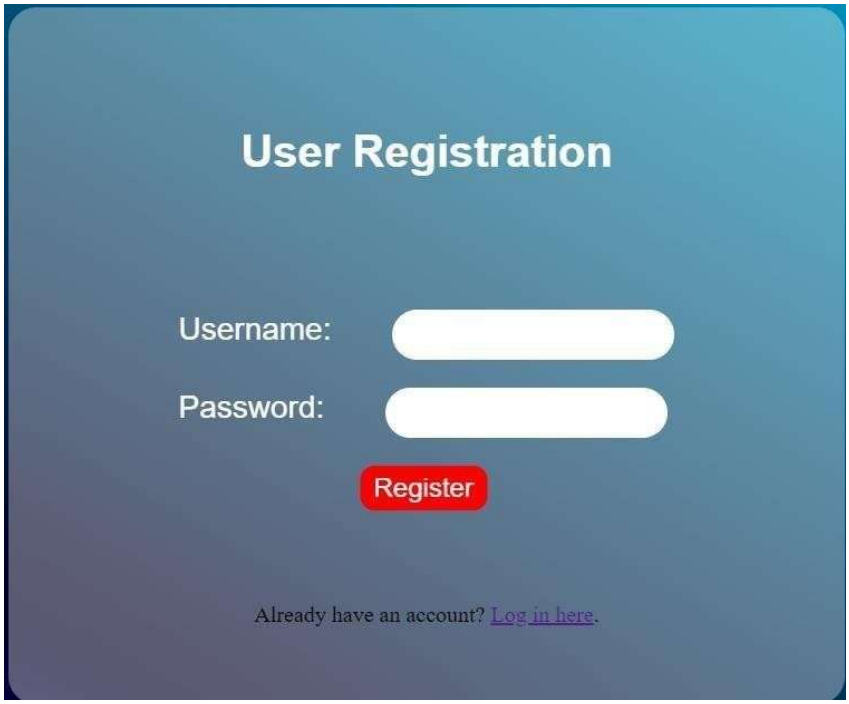
```
app.secret_key = secrets.token_hex(16)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(256), nullable=False)

with app.app_context():
    db.create_all()
@app.route("/")
def home():

return render_template("index.html")

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if User.query.filter_by(username=username).first():
            flash('Username already taken. Please choose another.', 'error')
        else:
            hashed_password = generate_password_hash(password, method='pbkdf2:sha256')
            new_user = User(username=username, password=hashed_password)
            db.session.add(new_user)
            db.session.commit()
            flash('Registration successful. You can now log in.', 'success')
            return redirect(url_for('login'))

    return render_template('register.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password, password):
            session['username'] = username
            flash('Login successful!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid username or password. Please try again.', 'error')

             return render_template('login.html')
```

```
    @app.route('/dashboard')
    def dashboard():
        if 'username' in session:
            return f'Welcome to the dashboard, {session["username"]}!'
        else:
            flash('Please log in to access the dashboard.', 'info')
            return redirect(url_for('login'))

    @app.route('/logout')
    def logout():
        session.pop('username', None)
        flash('You have been logged out.', 'info')
        return redirect(url_for('login'))

    if __name__ == '__main__':
app.run(debug=True)
```

Output:







Welcome to the dashboard, User1!

PRACTICAL- 3

- AIM: A program that creates a web application that allows users to upload and download files.

- CODE:

  - Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>File Upload and Download</title>
</head>
<body>
    <h1>File Upload and Download</h1>
    <form action="/upload" method="post" enctype="multipart/form-data">
        <label for="file">Choose a file:</label>
        <input type="file" name="file" id="file" required>
        <br>
        <input type="submit" value="Upload">
    </form>

    <h2>Uploaded Files</h2>
    {% for filename in filenames %}
        <div>
            <span>{{ filename }}</span>
            <a href="{{ url_for('download_file', filename=filename) }}" download>
                <button>Download</button>
            </a>
        </div>
    {% endfor %}
</body>
</html>
```

  - app.py

```
from flask import Flask, render_template, request, send_from_directory,
redirect, url_for
import os
app = Flask(_name_)
UPLOAD_FOLDER = 'uploads'
```

```
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


os.makedirs(UPLOAD_FOLDER, exist_ok=True)
@app.route('/')
def index():
    filenames = os.listdir(app.config['UPLOAD_FOLDER'])
    return render_template('index.html', filenames=filenames)
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return "No file part"
    file = request.files['file']
    if file.filename == '':
        return "No selected file"
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], file.filename))
    return redirect(url_for('index'))
@app.route('/download/<filename>')
def download_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)
if __name__ == '__main__':
        app.run(debug=True)
```

❖ Output:

PRACTICAL- 4

- AIM: A program that creates a web application that displays data from a database in a tabular format.

- CODE:

  - index.html

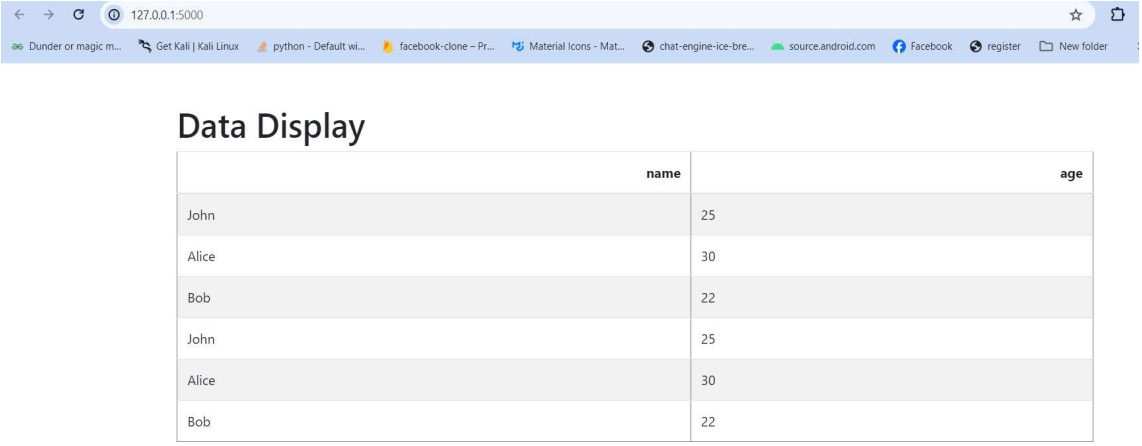
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Data Display</title>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
</head>
<body>
  <div class="container mt-5">
    <h1>Data Display</h1>
    <!-- Render the HTML table -->
    {{ table_html | safe }}
  </div>
</body>
</html>
```

  - app.py

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy
import pandas as pd

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Create a SQLAlchemy instance
db = SQLAlchemy(app)
```

```
# Define a model for the data
class Person(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    age = db.Column(db.Integer, nullable=False)

# Sample data for demonstration
sample_data = [{'name': 'John', 'age': 25},
               {'name': 'Alice', 'age': 30},
               {'name': 'Bob', 'age': 22}]

# Populate the database with sample data
with app.app_context():
    db.create_all()
    for entry in sample_data:
        person = Person(name=entry['name'], age=entry['age'])
        db.session.add(person)
db.session.commit()
# Define a route to display data in tabular format
@app.route('/')
def display_data():
    # Query data from the database
    data = Person.query.all()

    # Convert the data to a Pandas DataFrame
    df = pd.DataFrame([(person.name, person.age) for person in data], columns=['name', 'age'])

    # Convert the DataFrame to HTML for rendering in the template
    table_html = df.to_html(classes='table table-striped', index=False)

    return render_template('index.html', table_html=table_html)

if __name__ == '_main_':
    app.run(debug=True)
```

❖ Output:



name	age
John	25
Alice	30
Bob	22
John	25
Alice	30
Bob	22

PRACTICAL-5

- AIM: A program that creates a web application that accepts user input and sends it to a server-side script for processing.

- CODE:

  - Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>User Input</title>
  </head>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
    body {
      height: 100vh;
      width: 100%;
      background: #a2d2ff;
      display: flex;
      align-items: center;
      justify-content: center;
      flex-direction: column;
    }
    .container {
      display: flex;
      align-items: center;
      justify-content: space-evenly;
      flex-direction: column;
      width: 500px;
      height: 600px;
      border-radius: 20px;
      background: #ffffff5a;
      backdrop-filter: blur(20px);
      & h1{
        font-family: Arial, Helvetica, sans-serif;
```

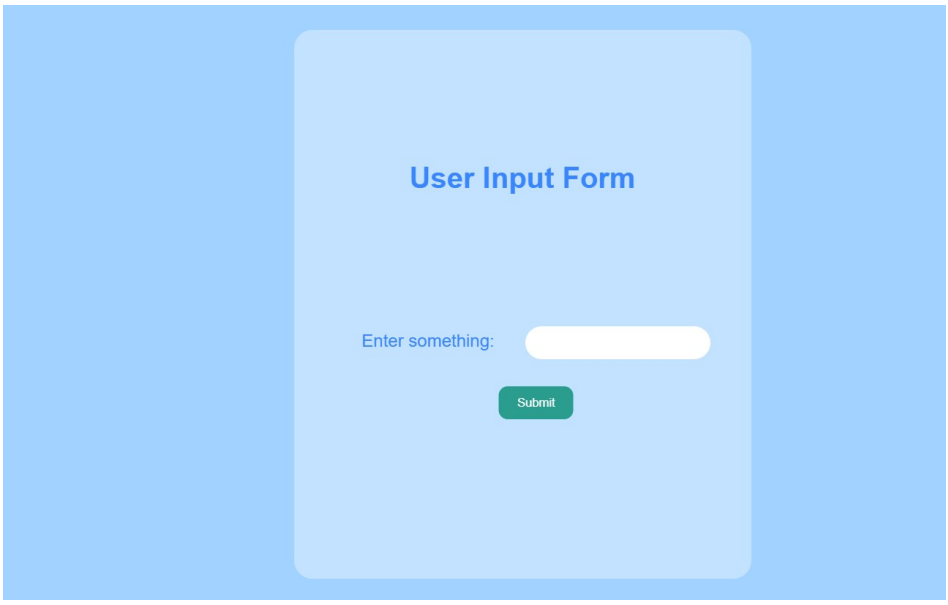
```
      color: #3a86ff;

    font-size: 2rem;
    }
    & label{
      color: #3a86ff;
      font-family: Arial, Helvetica, sans-serif;
      font-size: 1.2rem;
      padding: 10px;
      margin: 10px 20px;
    }
    & .enter{
      padding: 10px 20px;
      border: none;
      outline: none;
      border-radius: 20px;
    }
    & .submit{
      padding: 10px 20px;
      color: #fff;
      background: #2a9d8f;
      outline: none;
      border: none;
      border-radius: 10px;
      transition: .3s;
      transform: translateX(150px);
      margin: 30px;
      &:hover{
        color: #000;
        cursor: pointer;
        background: #fff;
      }
    }
    & h2{
      font-family: Arial, Helvetica, sans-serif;
      color: #3a86ff;
      font-size: 2rem;
    }
  }
</style>
<body>
  <div class="container">
    <h1>User Input Form</h1>
    <form method="post" action="/">
```

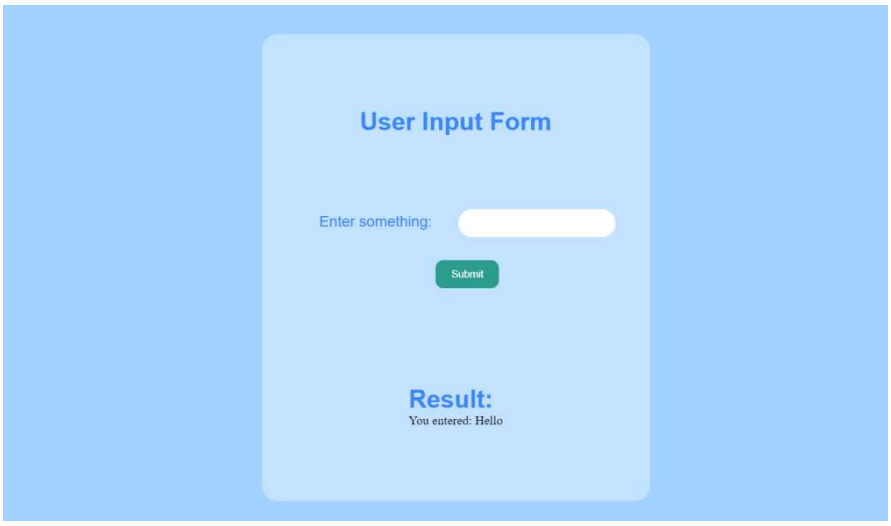
```
        <label for="user_input">Enter something:</label>
        <input type="text" class="enter" name="user_input" id="user_input" required />

          <br />
          <input class="submit" type="submit" value="Submit" />
        </form>

        {% if result %}
        <div>
          <h2>Result:</h2>
          <p>{{ result }}</p>
        </div>
        {% endif %}
      </div>
      </body>
</html>
```

- app.py

```
from flask import Flask, render_template, request
app = Flask(_name_)
# Define a route for the main page
@app.route('/', methods=['GET', 'POST'])
def index():
    result = None
    if request.method == 'POST':
        # Get user input from the form
        user_input = request.form.get('user_input')
        result = f"You entered: {user_input}"
    return render_template('index.html', result=result)

if__name__==' main ':
app.run(debug=True)
```

❖ Output:


User Input Form
Enter something:
Submit


User Input Form
Enter something:
Submit
Result:
You entered: Hello

SET-4

PRACTICAL-1

- AIM : A program that creates a web application that uses a template engine to generate dynamic HTML pages.
- CODE:
  - index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Flask Template Example</title>
</head>
<body>
<h1>{{ message }}</h1>
</body>
</html>
```

  - app.py

```
from flask import Flask, render_template
app = Flask( name )
```

```
@app.route("/")def home():

return render_template('index.html',message='Hello, World!')
if name == " main ":
app.run(debug=True)
```

Actual Output:

PRACTICAL- 2

- AIM: A program that creates a web application that supports AJAX requests and updates the page without reloading

- CODE:

  - index_ajax.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title> Flask AJAX Example </title>
<script>
async function updateMessage() {
const messageInput = document.getElementById('message');
const message = messageInput.value;
const response = await fetch('/update', {
method: 'POST',
headers: {
'Content-Type': 'application/json',
},
body: JSON.stringify({ 'message': message }),
});
const responseData = await response.json();
document.getElementById('output').innerHTML =
responseData.updatedMessage;
}
</script>
</head>
<body>
```

```
<h1>Flask AJAX Example</h1>

<input type="text" id="message" placeholder="Enter message">
<button onclick="updateMessage()">Update</button>
<div id="output"></div>
</body>
</html>
```

- app.py

```
from flask import Flask, render_template
app = Flask( name )
@app.route("/")def home():
return render_template('index.html',message='Hello, World!')
if name == " main ":
app.run(debug=True)
```

❖ **Output:**

PRACTICAL-3

- AIM: A program that creates a web application that uses Django's built-in debugging features to troubleshoot errors and exceptions.

- CODE:

  - manage.py

```
import os
import sys
if name == " main ":
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")
try:
from django.core.management import execute_from_command_line
except ImportError as exc:
raise ImportError(
"Couldn't import Django. Are you sure it's installed and "
"available on your PYTHONPATH environment variable? Did
you "
"forget to activate a virtual environment?"
) from exc
execute from command_line(sys.argv)
```

  - settings.py

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath( file )))
SECRET_KEY = 'your-secret-key'
DEBUG = True
ALLOWED_HOSTS = []
```

```
INSTALLED_APPS = [
'django.contrib.staticfiles',
]
MIDDLEWARE = [
'django.middleware.security.SecurityMiddleware',
]
ROOT_URLCONF = 'mysite.urls'
TEMPLATES = [
{
'BACKEND': 'django.template.backends.django.DjangoTemplates',
'DIRS': [os.path.join(BASE_DIR, 'templates')],
'APP_DIRS': True,
'OPTIONS': {
'context_processors': [
'django.template.context_processors.debug',
'django.template.context_processors.request',
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
],
},
},
]
WSGI_APPLICATION = 'mysite.wsgi.application'
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.sqlite3',
'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
}
}
```

```
STATIC_URL = '/static/'
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

- urls.py

```
from django.urls import path
from django.http import HttpResponseServerError
def trigger_error(request):
return HttpResponseServerError("Intentional Error for Debugging")
urlpatterns = [
path('error/', trigger_error),
]
```

❖ **Output:**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Lenovo> python manage.py
```

PRACTICAL- 4

- AIM: A program that creates a web application that implements user authentication and Authorization.

- CODE:

  - Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Static HTML Page</title>
  </head>
  <style>
    @import
   url("https://fonts.googleapis.com/css2?family=Poppins:wght@500&display=swap");
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
    body {
      height: 100vh;
      width: 100%;
      display: flex;
      justify-content: center;
      align-items: center;
      flex-direction: column;
      background: #ff5a5f;
    }
    h1 {
      font-family: "Poppins", sans-serif;
      color: #fff;
      margin: 30px 50px;
      font-size: 3rem;
    }
    input {
      padding: 10px 20px;
      border: 3px solid #fff;
      border-radius: 10px;
      background: rgb(16, 208, 16);
      font-size: 1.5rem;
      color: white;
```

```
            font-family: "Poppins", sans-serif;
            font-weight: 300;
            transition: .3s;
            &:hover{
            background: #fff;
            color: #000;
            cursor: pointer;
              }
          }
        </style>
        <body>
          <h1>Hello, this is a static HTML page served by Flask!</h1>
          <form action="{{ url_for('register') }}">
            <input type="submit" value="Register" />
          </form>
        </body>
    </html>
```

- login.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>User Login</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        height: 100vh;
        width: 100%;
        display: flex;
        align-items: center;
        justify-content: center;
        flex-direction: column;
        background: rgb(9, 9, 121);
        background: linear-gradient(
        30deg,
          rgba(9, 9, 121, 1) 0%,
          rgba(2, 0, 36, 1) 29%,
          rgba(0, 212, 255, 1) 100%
        );
      }
```

```
.container {
  display: flex;
  align-items: center;
  justify-content: space-evenly;
  flex-direction: column;
  width: 600px;
  border-radius: 20px;
  height: 500px;
  background: #ffffff5a;
backdrop-filter: blur(20px);
  & h1 {
    font-family: Arial, Helvetica, sans-serif;
    color: #fff;
    margin: 30px 0;
  }
  & li {
    list-style: none;
  }
  & form {
    & label {
      color: white;
      font-family: Arial, Helvetica, sans-serif;
      font-size: 1.4rem;
      margin: 10px 20px;
    }
    & .log_button {
      color: #fff;
      background: red;
      border: none;
      outline: none;
      padding: 5px 10px;
      border-radius: 10px;
      font-size: 1.2rem;
      transition: 0.3s;
      transform: translateX(130px);
      &:hover {
        background: #fff;
        color: #000;
        cursor: pointer;
      }
  }
  &  .password{
      padding: 10px 20px;
      border-radius: 20px;
      outline: none;
      border: none;
  }
  &  .username{
      padding: 10px 20px;
```

```
            border-radius: 20px;
            outline: none;
            border: none;
          }
          & input {
            margin: 10px 20px;
          }
          }
        }
        .error {
          color: red;
        }
        .success {
          color: green;
        }
        .default {
          color: black;
        }
      </style>
    </head>
    <body>
      <div class="container">
        <h1>User Login</h1>
        {% with messages = get_flashed_messages() %}  {% if messages %}
        <ul>
          {% for message in messages %}
          <li
            class="{% if 'error' in message %}error{% elif 'success' in message %}success{% else
      %}default{% endif
    %}"
          >
            {{ message }}
          </li>
          {% endfor %}
        </ul>
        {% endif %}  {% endwith %}
        <form method="post" action="{{ url_for('login') }}">
          <label for="username" class="username_label">Username:</label>
          <input type="text" name="username" class="username" required />
          <br />
          <label for="password" class="password_label">Password:</label>
          <input type="password" name="password" class="password" required />
          <br />
          <input type="submit" class="log_button" value="Log in" />
        </form>
        <p>
          Don't have an account?
          <a href="{{ url_for('register') }}">Register here</a>.
        </p>
```

```
    </div>
  </body>
</html>
```

- register.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>User Registration</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        height: 100vh;
        width: 100%;
        display: flex;
        align-items: center;
        justify-content: center;
        flex-direction: column;
        background: rgb(9, 9, 121);
        background: linear-gradient(
        30deg,
          rgba(9, 9, 121, 1) 0%,
          rgba(2, 0, 36, 1) 29%,
          rgba(0, 212, 255, 1) 100%
        );
      }
      .container {
        display: flex;
        align-items: center;
        justify-content: space-evenly;
        flex-direction: column;
        width: 600px;
        border-radius: 20px;
        height: 500px;
        background: #ffffff5a;
        backdrop-filter: blur(20px);
        & h1 {
          font-family: Arial, Helvetica, sans-serif;
          color: #fff;
```

```
    margin: 30px 0;
  }
  & li {
    list-style: none;
  }
  & form {
    & label {
      color: white;
      font-family: Arial, Helvetica, sans-serif;
      font-size: 1.4rem;
      margin: 10px 20px;
    }
    & .register_button {
      color: #fff;
      background: red;
      border: none;
      outline: none;
      padding: 5px 10px;
      border-radius: 10px;
      font-size: 1.2rem;
      transition: 0.3s;
      transform: translateX(130px);
      &:hover {
        background: #fff;
        color: #000;
        cursor: pointer;
      }
    }
    & .password {
      padding: 10px 20px;
      border-radius: 20px;
      outline: none;
      border: none;
    }
    & .username {
      padding: 10px 20px;
      border-radius: 20px;
      outline: none;
      border: none;
    }
    & input {
      margin: 10px 20px;
    }
  }
}
.error {
  color: red;
}
.success {
```

```
            color: green;
          }
          .default {
            color: black;
          }
        </style>
      </head>
      <body>
        <div class="container">
          <h1>User Registration</h1>
          {% with messages = get_flashed_messages() %}  {% if messages %}
          <ul>
            {% for message in messages %}
            <li
              class="{% if 'error' in message %}error{% elif 'success' in message %}success{% else
        %}default{% endif
      %}"
            >
              {{ message }}
            </li>
            {% endfor %}
          </ul>
          {% endif %}  {% endwith %}
          <form method="post" action="{{ url_for('register') }}">
            <label for="username" class="username_label">Username:</label>
            <input type="text" name="username" class="username" required />
            <br />
            <label for="password" class="password_label">Password:</label>
            <input type="password" name="password" class="password" required />
            <br />
            <input type="submit" class="register_button" value="Register" />
          </form>
          <p>
            Already have an account?
            <a href="{{ url_for('login') }}">Log in here</a>.
          </p>
        </div>
      </body>
    </html>
```

- app.py

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
import secrets

app = Flask(__name__)
app.secret_key = secrets.token_hex(16)
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(256), nullable=False)

with app.app_context():
    db.create_all()
@app.route("/")
def home():
    return render_template("index.html")

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if User.query.filter_by(username=username).first():
            flash('Username already taken. Please choose another.', 'error')
        else:
            hashed_password = generate_password_hash(password, method='pbkdf2:sha256')
            new_user = User(username=username, password=hashed_password)
            db.session.add(new_user)
            db.session.commit()
            flash('Registration successful. You can now log in.', 'success')
            return redirect(url_for('login'))

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password, password):
            session['username'] = username
            flash('Login successful!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid username or password. Please try again.', 'error')

    return render_template('login.html')
@app.route('/dashboard')
def dashboard():
```

```
    if 'username' in session:
        return f'Welcome to the dashboard, {session["username"]}!'
    else:
        flash('Please log in to access the dashboard.', 'info')
        return redirect(url_for('login'))

@app.route('/logout')
def logout():
    session.pop('username', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('login'))

if __name__ == '__main__':
app.run(debug=True)
```

❖ Output:

PRACTICAL- 5

- AIM: A program that creates a web application that integrates with third-party APIs to provide additional functionality.

- CODE:

  - Index_api.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
<title>Weather App</title>
</head>
<body>
<h1>Weather App</h1>
<form action="/weather" method="post">
<label for="city">Enter city:</label>
<input type="text" id="city" name="city" required>
<button type="submit">Get Weather</button>
</form>
</body>
</html>
```

- result.html

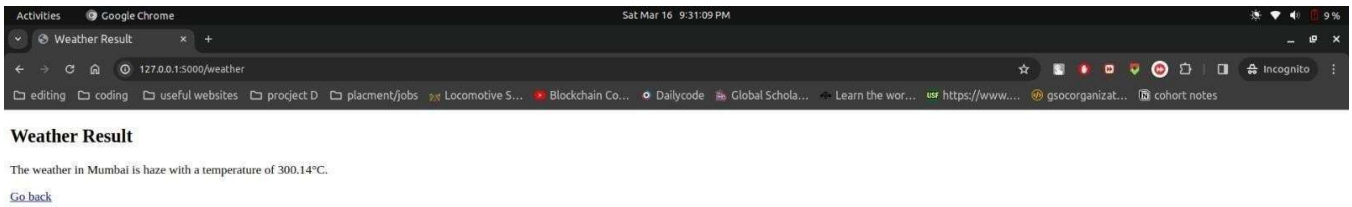
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
<title>Weather Result</title>
</head>
<body>
<h2>Weather Result</h2>
<p>{{ result }}</p>
<a href="/">Go back</a>
</body>
</html>
```

- app.py

```
from flask import Flask, render_template, request
import requests
app = Flask( name )
def get_weather(api_key, city):
url =
f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_k
ey}&units=metric'
response = requests.get(url)
data = response.json()
if response.status_code == 200:
```

```
weather_description = data['weather'][0]['description']
temperature = data['main']['temp']
return f'The weather in {city} is {weather_description} with a
temperature of {temperature}°C.'
else:
return 'Failed to fetch weather information.'
@app.route('/')
def home():
return render_template('index_api.html')
@app.route('/weather', methods=['POST'])
def weather():
api_key = 'your-openweathermap-api-key' # Replace with your API
key
city = request.form['city']
result = get_weather(api_key, city)
return render_template(s.html', result=result)
if name == ' main ':
app.run(debug=True)
```

❖ Output:



SET-5

PRACTICAL-1

- AIM : A program that creates a simple RESTful API that returns a list of users in JSON format

- CODE:

```
from flask import Flask, jsonify
app = Flask( name )
users = [
{'id': 1, 'name': 'Arshad'},
{'id': 2, 'name': 'Vishnu'},
{'id': 3, 'name': 'Reddy'}
]
@app.route('/users', methods=['GET'])
def get_users():
return  jsonify(users)
if name == ' main ':
app.run(debug=True)
```

- OUTPUT:

[{"id":1,"name":"John"},{"id":2,"name":"Jane"},{"id":3,"name":"Doe"}]

PRACTICAL- 2

- AIM: A program that creates a RESTful API that allows users to create, read, update, and delete resource

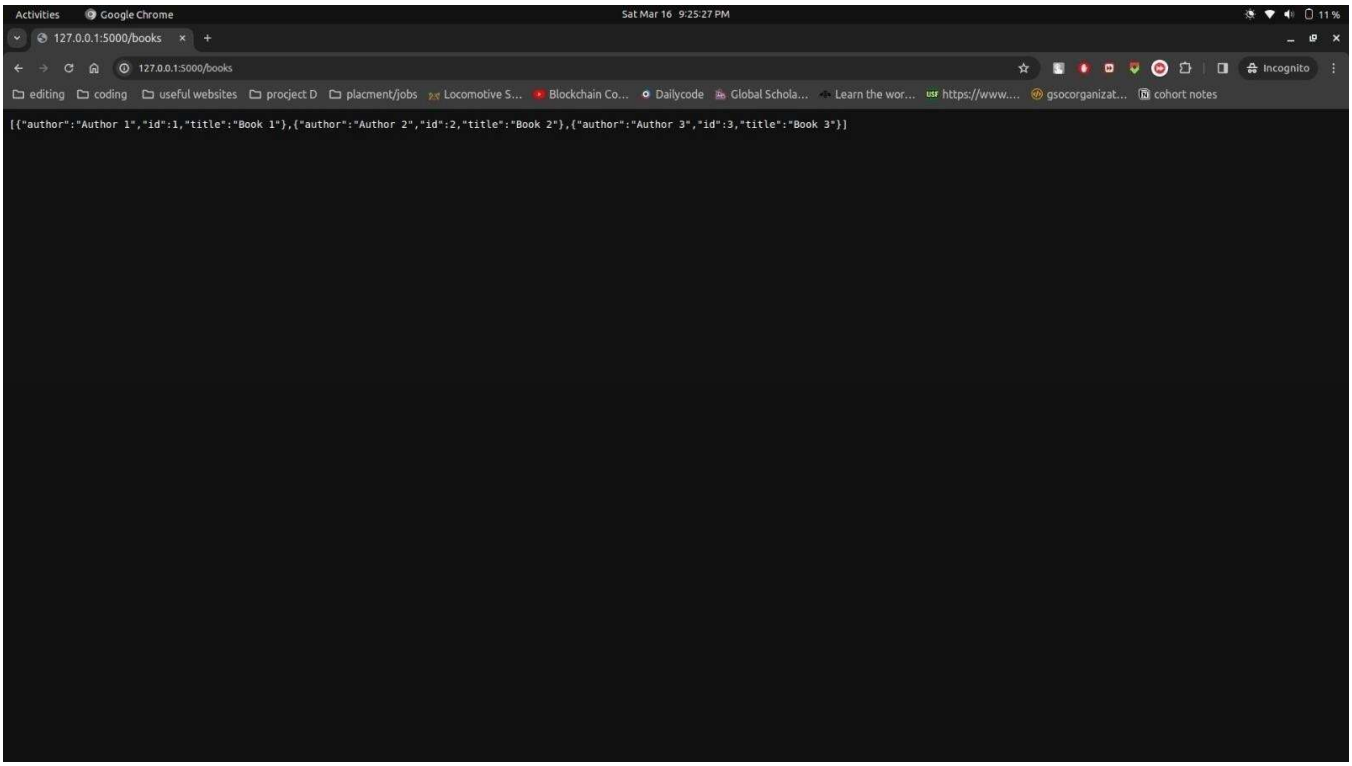
- CODE:

  - app.py

```
from flask import Flask, jsonify, request
app = Flask( name )
books = [
{'id': 1, 'title': 'Book 1', 'author': 'Author 1'},
{'id': 2, 'title': 'Book 2', 'author': 'Author 2'},
{'id': 3, 'title': 'Book 3', 'author': 'Author 3'}
]
@app.route('/books', methods=['GET'])
def get_books():
return jsonify(books)
@app.route('/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
book = next((b for b in books if b['id'] == book_id), None)
if book:
return jsonify(book)
else:
return jsonify({'error': 'Book not found'}), 404
@app.route('/books', methods=['POST'])
def create_book():
data = request.get_json()
new_book = {
```

```
'id': len(books) + 1,

'title': data['title'],
'author': data['author']
}
books.append(new_book)
return jsonify(new_book), 201
@app.route('/books/<int:book_id>', methods=['PUT'])
def update_book(book_id):
book = next((b for b in books if b['id'] == book_id), None)
if book:
data = request.get_json()
book['title'] = data['title']
book['author'] = data['author']
return jsonify(book)
else:
return jsonify({'error': 'Book not found'}), 404
@app.route('/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
global books
books = [b for b in books if b['id'] != book_id]
return jsonify({'result': True})
if name == ' main ':
app.run(debug=True)
```

❖ OUTPUT:


Activities
Google Chrome
Sat Mar 16 9:25:27 PM
127.0.0.1:5000/books
127.0.0.1:5000/books
Incognito
editing
coding
useful websites
project D
placment/jobs
Locomotive S...
Blockchain Co...
Dailycode
Global Schola...
Learn the wor...
https://www....
gsocorganizat...
cohort notes
[{"author":"Author 1","id":1,"title":"Book 1"},{"author":"Author 2","id":2,"title":"Book 2"},{"author":"Author 3","id":3,"title":"Book 3"}]

PRACTICAL-3

- AIM: A program that creates a RESTful API that authenticates users using a JSON Web Token

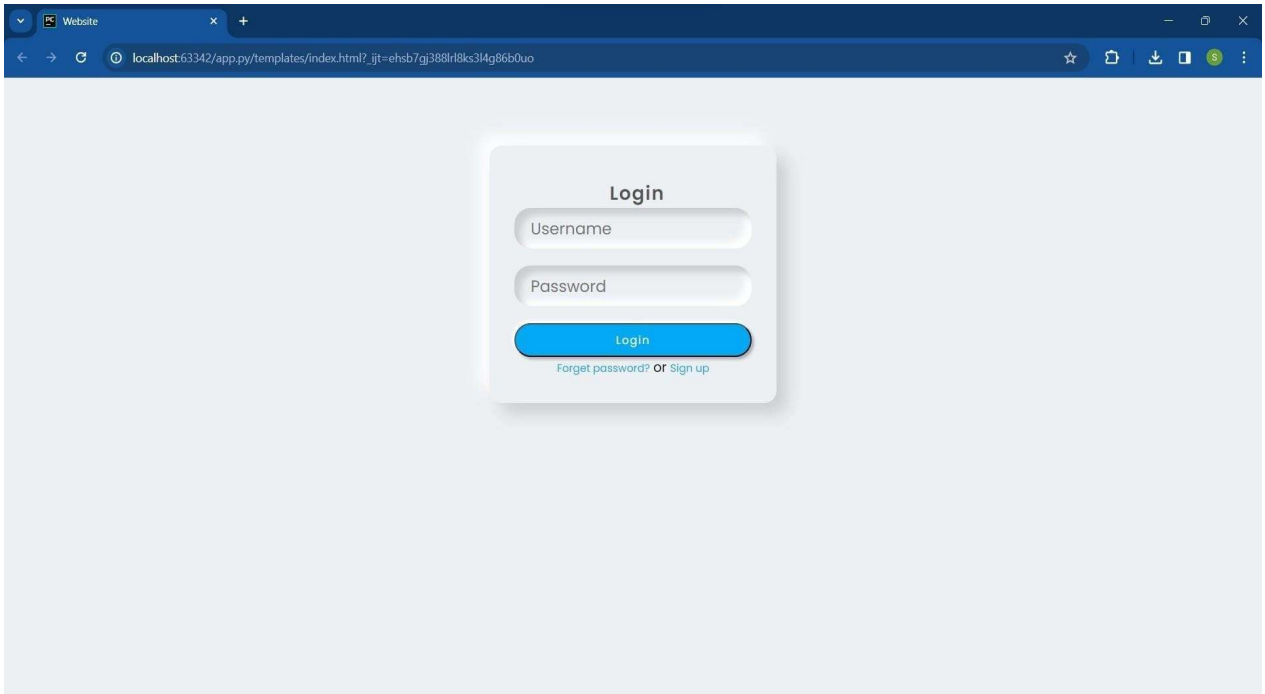
- CODE:

  - app.py

```
from flask import Flask, jsonify, request
from flask_jwt_extended import JWTManager, jwt_required,
create_access_token
app = Flask( name )
# Set up Flask-JWT-Extended
app.config['JWT_SECRET_KEY'] = 'your-secret-key' # Replace with your
secret key
jwt = JWTManager(app)
# Dummy user data (replace with a proper user database in a real
application)
users = {
'user1': {'password': 'password1'},
'user2': {'password': 'password2'}
}
# Route to generate a JWT token upon login
@app.route('/login', methods=['POST'])
def login():
data = request.get_json()
username = data.get('username')
password = data.get('password')
```

```
if username in users and users[username]['password'] == password:
access_token = create_access_token(identity=username)
return jsonify(access_token=access_token)
else:
return jsonify({'error': 'Invalid username or password'}), 401
# Protected route that requires a valid JWT token for access
@app.route('/protected', methods=['GET'])
@jwt_required()
def protected():
current_user = jwt.get_jwt_identity()
return jsonify(logged_in_as=current_user), 200
if name  == ' main ':
app.run(debug=True)
```

❖ OUTPUT:



PRACTICAL-4

- AIM: A program that creates a RESTful API that paginates the results of a query to improve performance

- CODE:

  - app.py

```
from flask import Flask, jsonify, request
app = Flask( name )
# Dummy data (replace with your actual data source)
items = [f'Item {i}' for i in range(1, 101)]
# Route that supports pagination
@app.route('/items', methods=['GET'])
def get_items():
page = int(request.args.get('page', 1))
per_page = int(request.args.get('per_page', 10))
start_index = (page - 1) * per_page
end_index = start_index + per_page
paginated_items = items[start_index:end_index]
return jsonify({'items': paginated_items, 'page': page,
'per_page': per_page, 'total_items': len(items)})
if name == ' main ':
    app.run(debug=True)
```

❖ OUTPUT:

127.0.0.1:5000/items?

{"items":["Item 1","Item 2","Item 3","Item 4","Item 5","Item 6","Item 7","Item 8","Item 9","Item 10"],"page":1,"per_page":10,"total_items":100}

127.0.0.1:5000/items?page=2&per_page=20

{"items":["Item 21","Item 22","Item 23","Item 24","Item 25","Item 26","Item 27","Item 28","Item 29","Item 30","Item 31","Item 32","Item 33","Item 34","Item 35","Item 36","Item 37",
38","Item 39","Item 40"],"page":2,"per_page":20,"total_items":100}

PRACTICAL- 5

- AIM: A program that creates a RESTful API that supports data validation and error handling.

- CODE:

  - app.py

```
from flask_restful import Resource, Api, reqparse
app = Flask( name )
api = Api(app)
# Dummy data (replace with your actual data source)
items = {'1': {'name': 'Item 1', 'price': 10.99},
'2': {'name': 'Item 2', 'price': 19.99}}
# Request parser for input validation
parser = reqparse.RequestParser()
parser.add_argument('name', type=str, required=True, help='Name cannot
be blank')
parser.add_argument('price', type=float, required=True, help='Price
cannot be blank')
class ItemResource(Resource):
def get(self, item_id):
item = items.get(item_id)
if item:
return item
else:
return {'error': 'Item not found'}, 404
def put(self, item_id):
args = parser.parse_args()
```

```
items[item_id] = {'name': args['name'], 'price':
args['price']}
return items[item_id], 201
def delete(self, item_id):
if item_id in items:
del items[item_id]
return {'result': True}
else:
return {'error': 'Item not found'}, 404
api.add_resource(ItemResource, '/items/<item_id>')
if name == ' main ':
    app.run(debug=True)
```

❖ OUTPUT:

{"name": "Item 1", "price": 10.99}