Parul University

FACULTY OF ENGINEERING AND TECHNOLOGY
BACHELOR OF TECHNOLOGY

# PROGRAMMING IN PYHTON WITH FULL STACK DEVELOPMENT LABORATORY (303105258)

IV SEMESTER

Computer Science & Engineering Department

Parul® University NAAC A++ ACCREDITED UNIVERSITY

Laboratory Manual
Session 2024-25

# CERTIFICATE

This is to certify that

Mr./Ms.....

with enrolment no. ..... has successfully completed his/her laboratory experiments in the **PPFSD-LAB (303105258)** from the department of **COMPUTER SCIENCE & ENGINEERING** during the academic year 2024-2025.

Date of Submission:.....  Staff In charge:.....

Head Of Department:.....

# TABLE OF CONTENT

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **14.** | A program that reads a CSV file and calculates the average of the values in a specified column. | | | | | | |
| **15.** | A program that reads an Excel file and prints the data in a tabular format. | | | | | | |
| | SET-3 | | | | | | |
| **16.** | A program that creates a simple web server and serves a static HTML page. | | | | | | |
| **17.** | A program that creates a web application that allows users to register and login. | | | | | | |
| **18.** | A program that creates a web application that allows users to upload and download files | | | | | | |
| **19.** | A program that creates a web application that displays data from a database in a tabular format. | | | | | | |
| **20.** | A program that creates a web application that accepts user input and sends it to a server-side script for processing. | | | | | | |
| | SET-4 | | | | | | |
| **21.** | A program that creates a web application that uses a template engine to generate dynamic HTML pages. | | | | | | |
| **22.** | A program that creates a web application that supports AJAX requests and updates the page without reloading. | | | | | | |
| **23.** | A program that creates a web application that uses Django's built-in debugging features to troubleshoot errors and exceptions. | | | | | | |
| **24.** | A program that creates a web application that implements user authentication and authorization. | | | | | | |
| **25.** | A program that creates a web application that integrates with third-party APIs to provide additional functionality. | | | | | | |

| SET-5 | | | | | | | |
|---|---|---|---|---|---|---|---|
| **26.** | A program that creates a simple RESTful API that returns a list of users in JSON format. | | | | | | |
| **27.** | A program that creates a RESTful API that allows users to create, read, update, and delete resources. | | | | | | |
| **28.** | A program that creates a RESTful API that authenticates users using a JSON Web Token. | | | | | | |
| **29.** | A program that creates a RESTful API that paginates the results of a query to improve performance | | | | | | |
| **30.** | A program that creates a RESTful API that supports data validation and error handling. | | | | | | |

# SET-1

## PRACTICAL-1

- AIM : A program that converts temperatures from Fahrenheit to Celsius and vice versa.
- CODE:

```
celsius =int(input("\n"))

fahrenheit = (celsius * 1.8) + 32

print(celsius,"celsius is equal to",fahrenheit, "fahrenheit")


fahrenheit2 =int(input("\n"))

celsius2=  (fahrenheit-32)*1.8

print(fahrenheit2,"fahrenheit is equal to",celsius2, "celsius")
```

- **Output:**

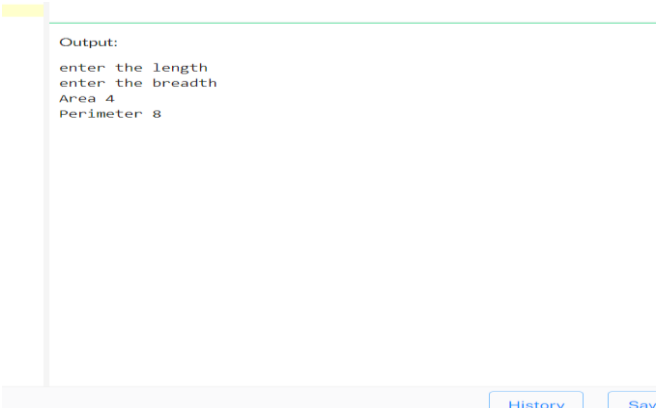
# PRACTICAL- 2

- AIM: A program that calculates the area and perimeter of a rectangle.

- CODE:

```
length=int(input("enter the length\n"))
breadth=int(input("enter the  breadth\n"))
Area=length*breadth
Perimeter = 2*(length+breadth)
print("Area",Area)
print("Perimeter",Perimeter)
```

- CODE:



# PRACTICAL- 3

- AIM: A program that generates a random password of a specified length.

- CODE:

```
import string
import random
def generate(n):
  c=string.ascii_letters +string.digits + string.punctuation
  password=''.join(random.choice(c) for _ in range(n))
  return password
n=int(input("enter the length of the password\n"))
r=generate(n)
print(r)
```

- OUTPUT:

# PRACTICAL- 4

- AIM: A program that calculates the average of a list of numbers.

- CODE:

```
list=[1,2,4,5,7,8]
average=sum(list)/len(list) print("average
of list elements:",average
```

- OUTPUT

```
Output:

average of list elements: 4.5
```

# PRACTICAL- 5

- AIM: A program that checks if a given year is a leap year
- CODE:

```
year=int(input("\n"))
if year%4==0 and year%100!=0 or year%400==0 :
  print(year,"is a leap year")
else:
    print(year,"is not a leap year")
```

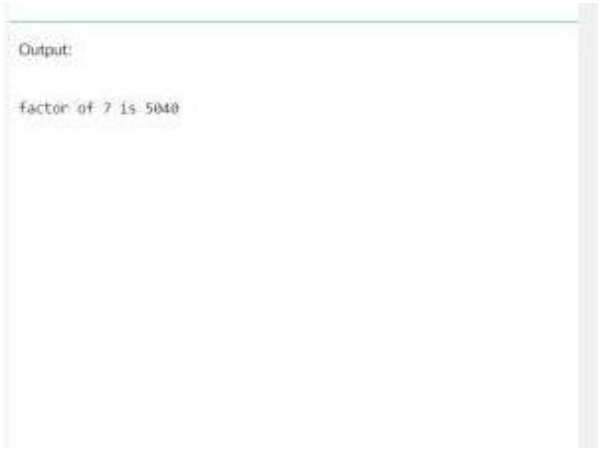
- OUTPUT:

# PRACTICAL- 6

- AIM: A program that calculates the factorial of a number
- CODE:

```
h=int(input("\n"))
fact=1
for i in range(1,h+1):
 fact=fact*i
print("factor of",h,"is",fact)
```

- OUTPUT:

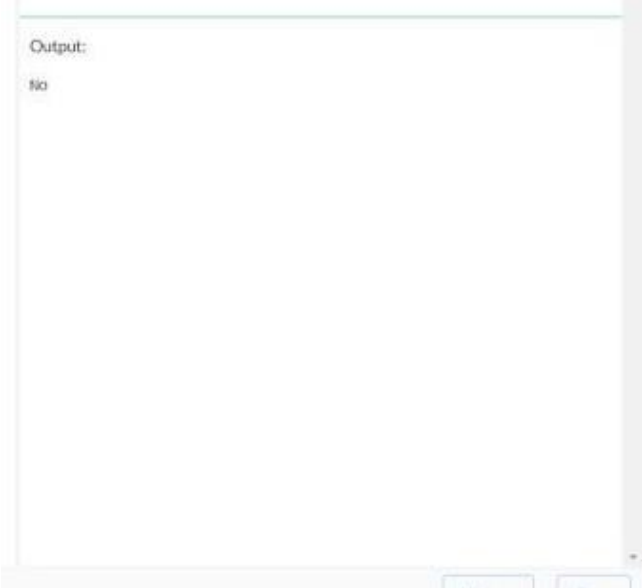



# PRACTICAL- 7

- AIM: A program that checks if a given string is a palindrome
- CODE:

```
def isPalindrome(s):
        return s == s[::-1]
s = "car"
ans = isPalindrome(s)
if ans:
        print("Yes")
else:
print("No")
```

❖ OUTPUT:



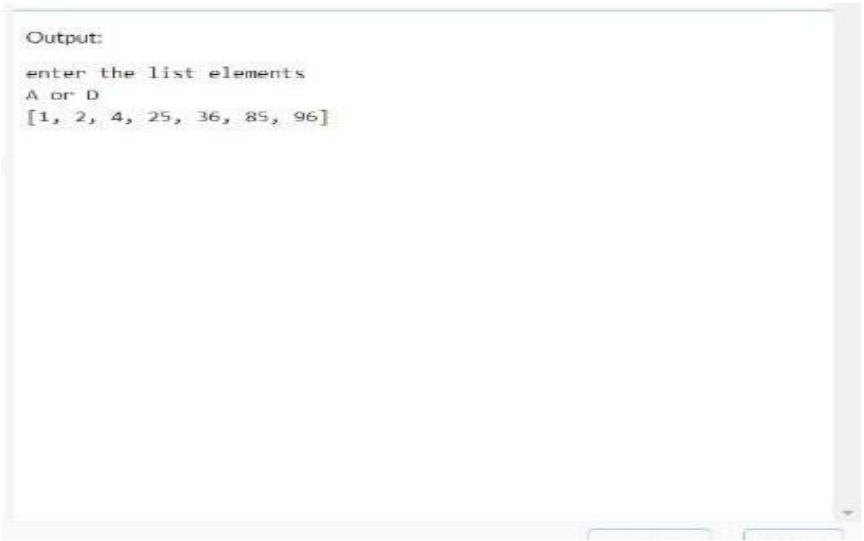

## PRACTICAL- 8

❖ AIM: A program that sorts a list of numbers in ascending or descending order

❖ CODE:

```
def sort():
  if    order=="A":
  Sl=sorted(list1)
  elif order=="D":
    Sl=sorted(list1,reverse=True)
  else:
    print("involid")
    return
  print(Sl)
list1=input("enter the list elements\n").split()
list1=[int(n) for n in list1]
order=input("A  or  D\n")
sort()
```

❖ OUTPUT:



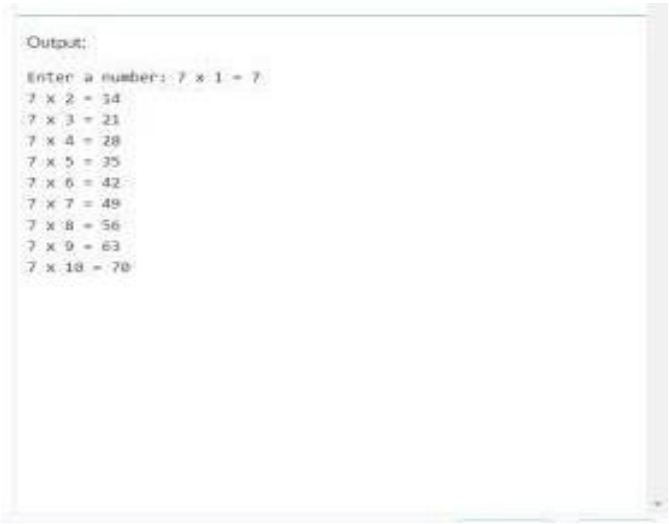

## PRACTICAL- 9

❖ AIM: A program that generates a multiplication table for a given number.

❖ CODE:

```
n = int(input("Enter a number: "))
for i in range(1,11):
  print(f"{n} x {i} = {n*i}")
```

❖ OUTPUT:



# PRACTICAL- 10

- AIM: A program that converts a given number from one base to another.
- CODE:

```
def  decimal_others(value,choice):
 if choice==1:
 return value
 elif  choice==2:
 return '{0:b}'.format(value)
 elif choice==3:
 return '{0:o}'.format(value)
 elif choice==4:
 return '{0:x}'.format(value)
 else:
 return "Invalid Option"
def binary_others(value,choice):
 if choice==1:
 return value
 elif  choice==2:
 return int(value,2)
 elif choice==3:
 return '{0:o}'.format(int(value,2))
 elif choice==4:
 return '{0:x}'.format(int(value,2))
 else:
 return "Invalid Option"
def octal_others(value,choice):
 if choice==1:
 return value
 elif  choice==2:
 return int(value,8)
 elif choice==3:
 return '{0:b}'.format(int(value,8))
 elif choice==4:
 return '{0:x}'.format(int(value,8))
 else:
```

```
 return "Invalid Option"
def hex_others(value,choice):
 if choice==1:
 return value
 elif  choice==2:
 return int(value,16)
 elif choice==3:
 return '{0:0}'.format(int(value,16))
 elif choice==4:
 return '{0:b}'.format(int(value,16))
 else:
 return "Invalid Option"
print("Convert from: 1: decimal ,2: binary,3: octal 4:hexadecimal")
input_choice=int(input("Enter the choice"))
if    input_choice==1:
 decimal_num=int(input("Enter decimal number"))
 print('Convert to: 1: decimal ,2: binary,3: octal 4:hexadecimal')
choice=int(input("Enter Target conversion:\n"))
print("Converted value: ",decimal_others(decimal_num,choice))
elif input_choice==2:
 binary_num=input("Enter  decimal  number")
 print('Convert to: 1: binary ,2: decimal,3: octal 4:hexadecimal')
choice=int(input("Enter Target conversion:\n"))
print("Converted value: ",binary_others(binary_num,choice))
elif input_choice==3:
 octal_num=input("Enter  decimal  number")
 print('Convert to: 1: octal ,2: decimal,3: binary 4:hexadecimal')
 choice=int(input("Enter Target conversion:\n"))
 print("Converted value: ",octal_others(octal_num,choice))
elif input_choice==4:
 hex_num=input("Enter decimal number")
 print('Convert to: 1: hex,2: decimal,3: octal 4:binary')
choice=int(input("Enter Target conversion:\n"))
print("Converted value: ",hex_others(hex_num,choice))
```

## ❖ OUTPUT:

STDIN

```
1
23
2
```

Output:

```
Convert from:
1: decimal
2: binary
3: octal
4: hexadecimal
 5:Exit

Enter the choice
Enter decimal numberConvert to:
1: decimal ,
2: binary
3: octal
4: hexadecimal

Enter Target conversion:

Converted value:  10111
Convert from:
1: decimal
2: binary
3: octal
4: hexadecimal
 5:Exit
```

# SET-2

## PRACTICAL-1

- AIM: 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 ": # Create a bank
my_bank = Bank("My Bank") customer_list=[]
while True:
print("1. New Customer 2. Existing Customer 3. Find Customers info 4.Exit") try:
choice = int(input())

if choice==1:
print("Customer Registration: \n") # Create a customer
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 Saving account 2. To Create Cheking 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")

if choice==2:
# User input for transactions
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("1. Enter 1 to deposit\n2. Enter 2 to Withdrawl\n3. Enter 3 to Check the Balance\n4. Exit")
   option=int(input("Enter your Option:\n"))
```

```
if option==1:
print("Welcome to Deposit Section\n") # Deposit
deposit_amount = int(input("\nEnter the amount to deposit: INR "))
   account_to_transact.deposit(deposit_amount)
elif option==2:
print("Welcome to withdrawl section:\n") # Withdrawal
withdrawal_amount = int(input("\nEnter the amount to withdraw: INR "))
   account_to_transact.withdraw(withdrawal_amount)
elif option==3:
# Display updated account information print("\nUpdated Account Information:") print(account_to_transact)
elif option==4: break
else:
print("Invalid Option")
else:
print("Account not found.") if choice==3:
my_bank.display_bank_info() elif choice==4:
break
else:
pass
except ValueError:
print("Invalid input. Please enter a valid option.")
continue
```

## ❖ OUTPUT:

STDIN

```
1
Ronak Parmar
Vadodara
9601264186
1
3
4
```

Output:

```
1. New Customer 2. Existing Customer 3. Find Customers info 4.Exit
Customer Registration:

Enter Customer Name:Enter Customer Address: Enter Customer Contact Number
Savings account created with account number: 95699603

1. New Customer 2. Existing Customer 3. Find Customers info 4.Exit
Bank Name: My Bank
Customers:
Customer Name: Ronak Parmar
Address: Vadodara
Contact Number: 9601264186
Accounts:
  - Savings Account - Account Number: 95699603, Balance: INR 1000

1. New Customer 2. Existing Customer 3. Find Customers info 4.Exit
```

# PRACTICAL- 2

- AIM: 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 = []
print("""1.Student_form/details 2.Teacher_form/details
3. Course_form/details""")
cho = int(input("\nEnter your choice: "))

if cho == 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"\nEnter student {i +
  1} name: ") grade = input(f"\nEnter student {i + 1} grade: ")
  students.append(Student(student_id, name, grade)) print("\nRegistration successful.")
elif cho == 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"\nEnter teacher {i +
  1} name: ") subject = input(f"\nEnter teacher {i + 1} subject: ")
  teachers.append(Teacher(teacher_id, name, subject))

print("\nRegistration successful.")
elif cho == 3:
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"\nEnter
  course {i + 1} name: ")
teacher_index = int(input("\nEnter the index of the teacher for this course: ")) teacher =
  teachers[teacher_index]
student_indices = input("\nEnter the indices of students for this course (comma-
  separated): ") student_indices = student_indices.split(",")
```

```
students_for_course = [students[int(index)] for index in student_indices]
  courses.append(Course(course_code, course_name, teacher, students_for_course))
  print("\nRegistration successful.")
else:
print("\nInvalid input")
if  name        == " main ":
 main()
```

❖ OUTPUT:

STDIN

```
1
1
34090394
Ronak Parmar
80
```

Output:

```
1.Student_form/details
2.Teacher_form/details
3.Course_form/details

Enter your choice:
Enter the number of students:
Enter student 1 ID:
Enter student 1 name:
Enter student 1 grade:
Registration successful.
```

# PRACTICAL- 3

- AIM: 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 valid file path."
path ="example.txt" print(count(path))
```

- OUTPUT:

Output:

```
data = ['Parul', 'University,', 'Be', 'here,', 'Be', 'vibrant.']
length of the words: 6
```

# PRACTICAL- 4

- AIM: 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 column_name not in reader.fieldnames:
print(f"Column '{column_name}' not found in the CSV file.") return None
total = 0
count = 0
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"File '{csv_file}' not found.") 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}")
```

❖ OUTPUT:

Output:

```
Skipping row 14: Invalid value in column 'CN'.
The average value in column 'CN' is: 58.363636363333335
```

# PRACTICAL- 5

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

- CODE:

```
import pandas as pd import openpyxl
output = pd.read_excel("delimited.xlsx")
print(output)
```

- OUTPUT:

Output:

| | Sr No. | Name | Enrollment | MATHS | CN | OS | PFSD | CC |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | Rakesh | 2203051240086 | 80 | 4 | 44 | 80 | 22 |
| 1 | 2 | Ritesh | 2203051240112 | 99 | 45 | 77 | 70 | 55 |
| 2 | 3 | Rohit | 2203051240089 | 3 | 71 | 23 | 44 | 71 |
| 3 | 4 | Rutal | 2203051240124 | 33 | 44 | 23 | 70 | 3 |
| 4 | 4 | Gautam | 2203051240096 | 40 | 4 | 33 | 44 | 3 |
| 5 | 5 | Pritesh | 2203051240080 | 80 | 34 | 23 | 80 | 34 |
| 6 | 6 | Raju | 2203051249002 | 20 | 22 | 20 | 34 | 20 |
| 7 | 7 | Ramesh | 2203051240094 | 50 | 2 | 22 | 32 | 50 |
| 8 | 8 | Sudeep | 2203051240121 | 22 | 11 | 50 | 3 | 50 |
| 9 | 9 | Sanjay | 2203051240125 | 11 | 4 | 81 | 44 | 81 |
| 10 | 10 | Jethalal | 2203051240115 | 44 | 33 | 80 | 53 | 23 |

# 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>
```

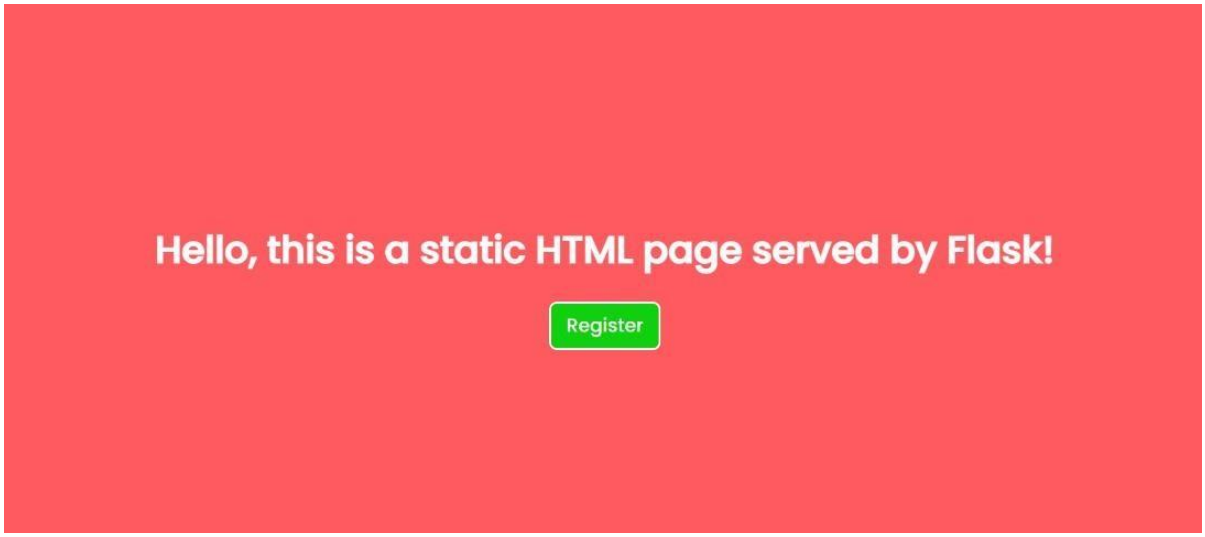
```
        {% with messages = get_flashed_messages() %} {% if messages %}
        <ul>
          {% for message in messages %}
          <li
            class="{% if 'error' in message %}error{% elif 'success' in message %}success{% else
      %}default{% endif
    %}"
          >
            {{ message }}
          </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>
```

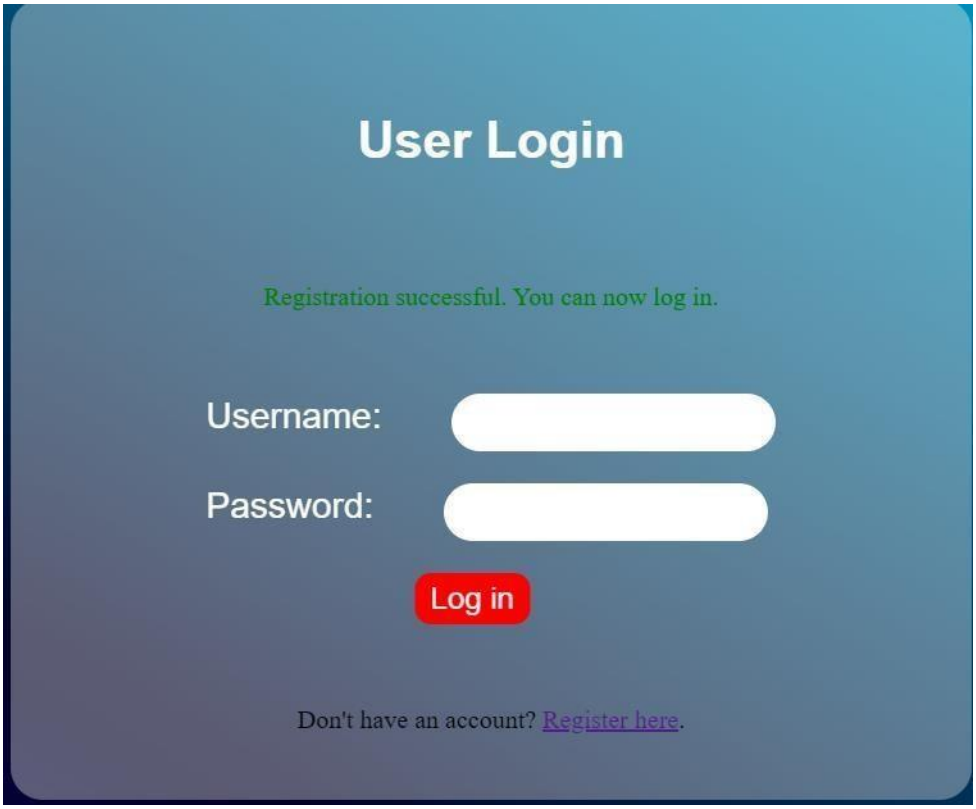
- app.py

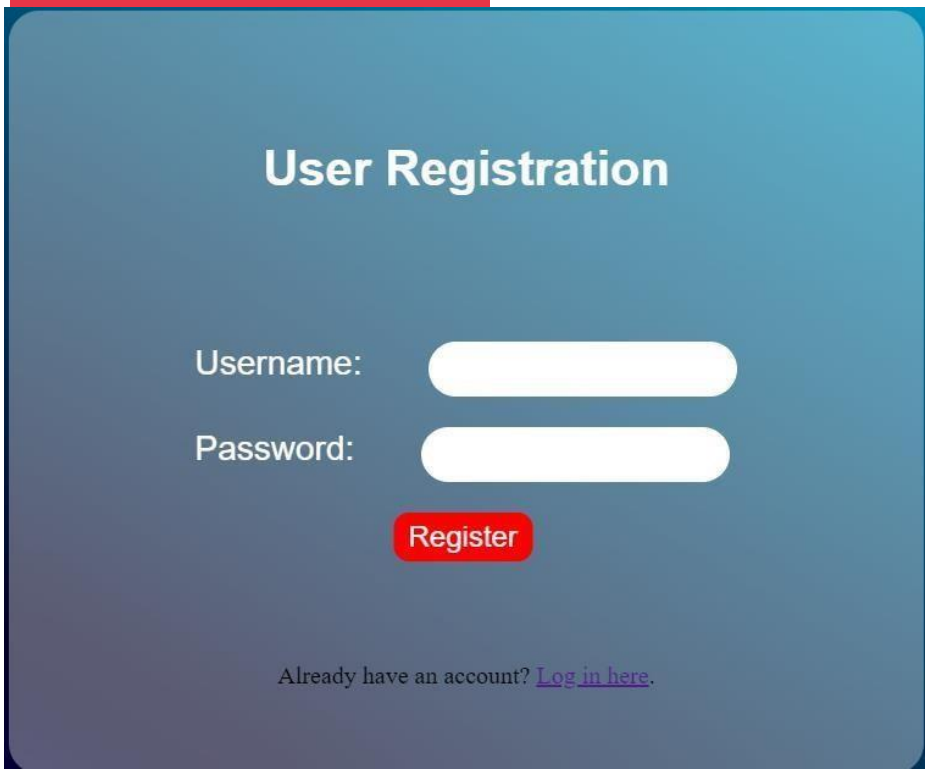
```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
import secrets

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:


Hello, this is a static HTML page served by Flask!
Register


User Login
Registration successful. You can now log in.
Username:
Password:
Log in
Don't have an account? Register here.




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
```

18

```
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

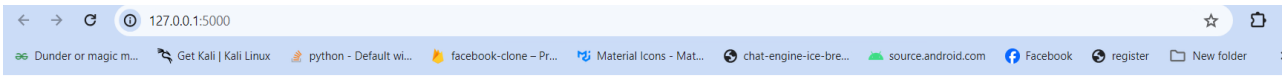
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Data Display</title>
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
</head>
<body>
   <div class="container 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:




# Data Display

| 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;
```

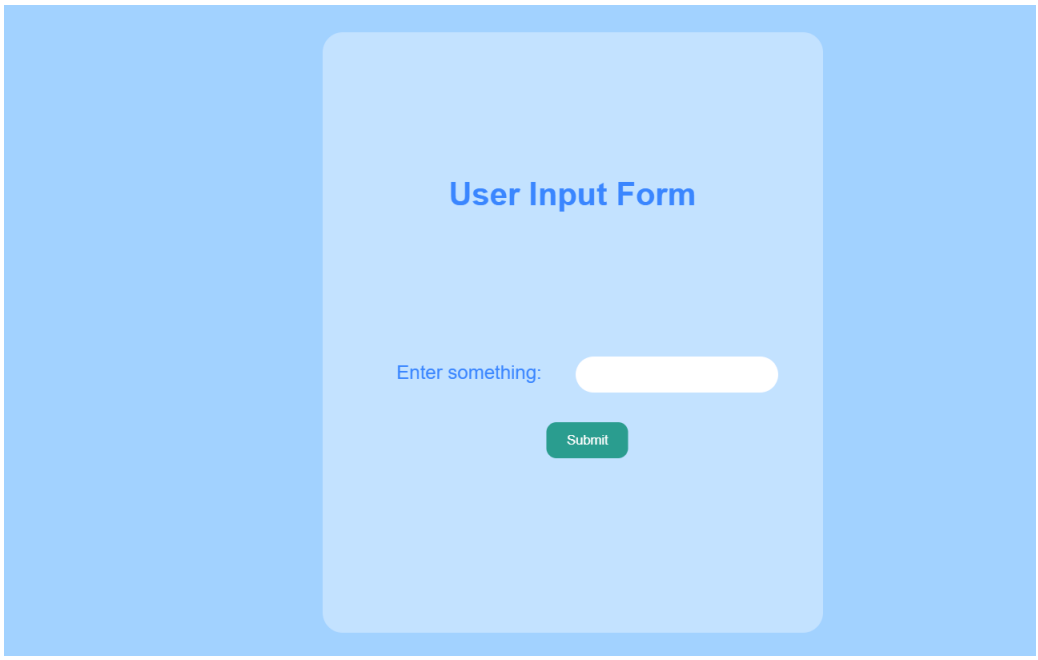
```
 font-size: 2rem;
   }
   & label{
     color: #3a86ff;
     font-family: Arial, Helvetica, sans-serif;
     font-size: 1.2rem;
     padding: 10px;
     margin: 10px 20px;
   }
   & .enter{
     padding: 10px 20px;
     border: none;
     outline: none;
     border-radius: 20px;
   }
   & .submit{
     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 />
```

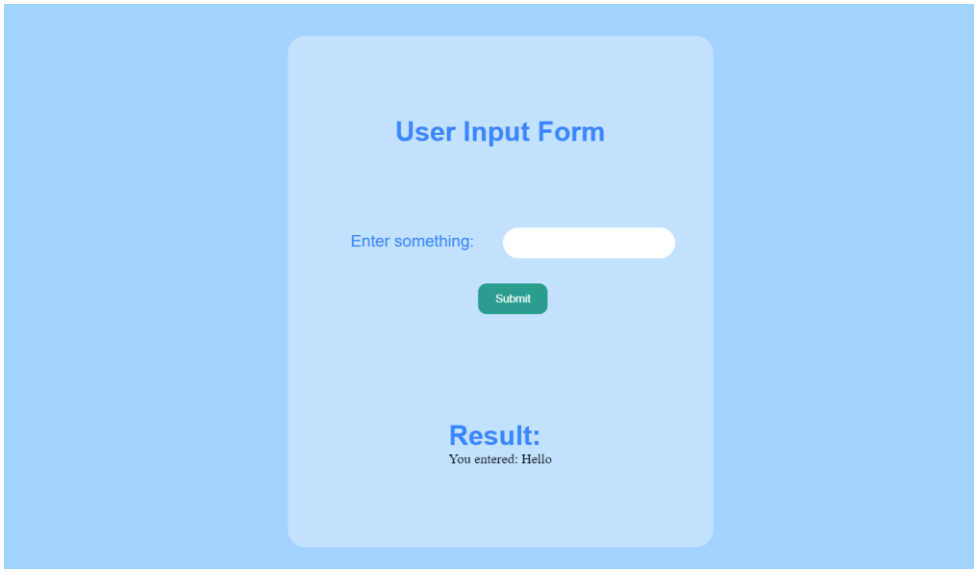
```
        <br />
        <input class="submit" type="submit" value="Submit" />
      </form>

      {% 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:


**Hello, World!**

# PRACTICAL-2

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

- CODE:

  - index_ajax.html

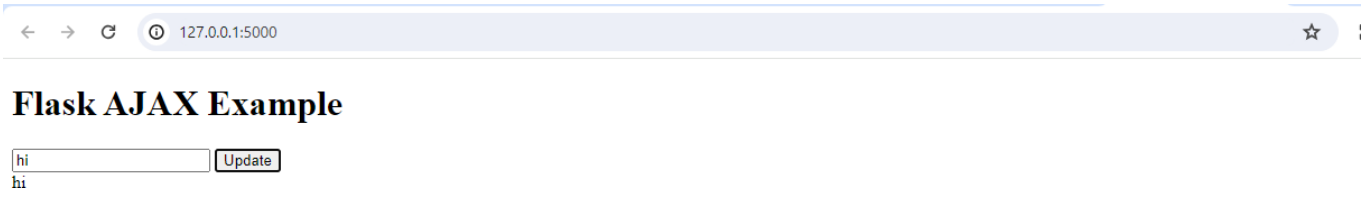
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title> Flask AJAX Example </title>
<script>
async function updateMessage() {
const messageInput = document.getElementById('message');
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 = []
```

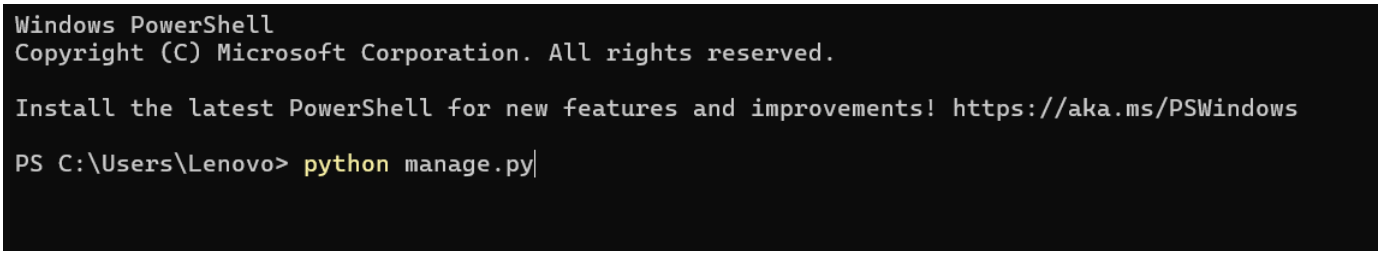
```
INSTALLED_APPS = [
'django.contrib.staticfiles',
]
MIDDLEWARE = [
'django.middleware.security.SecurityMiddleware',
]
ROOT_URLCONF = 'mysite.urls'
TEMPLATES = [
{
'BACKEND': 'django.template.backends.django.DjangoTemplates',
'DIRS': [os.path.join(BASE_DIR, 'templates')],
'APP_DIRS': True,
'OPTIONS': {
'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:**



# 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

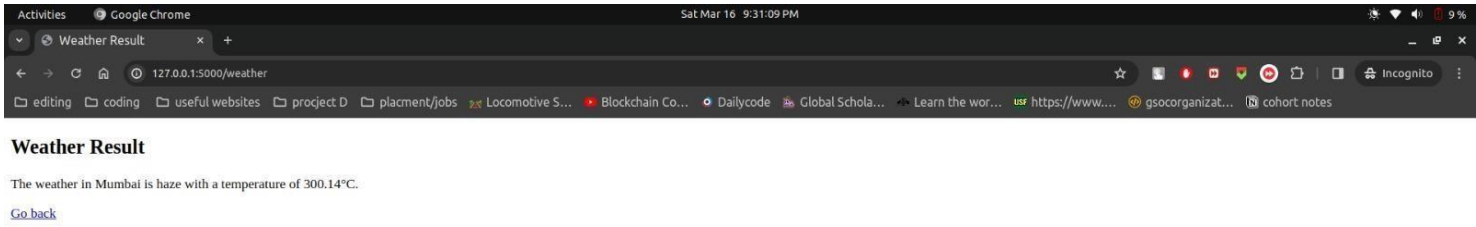
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
<title>Weather Result</title>
</head>
<body>
<h2>Weather Result</h2>
<p>{{ result }}</p>
<a href="/">Go back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

# PRACTICAL- 2

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

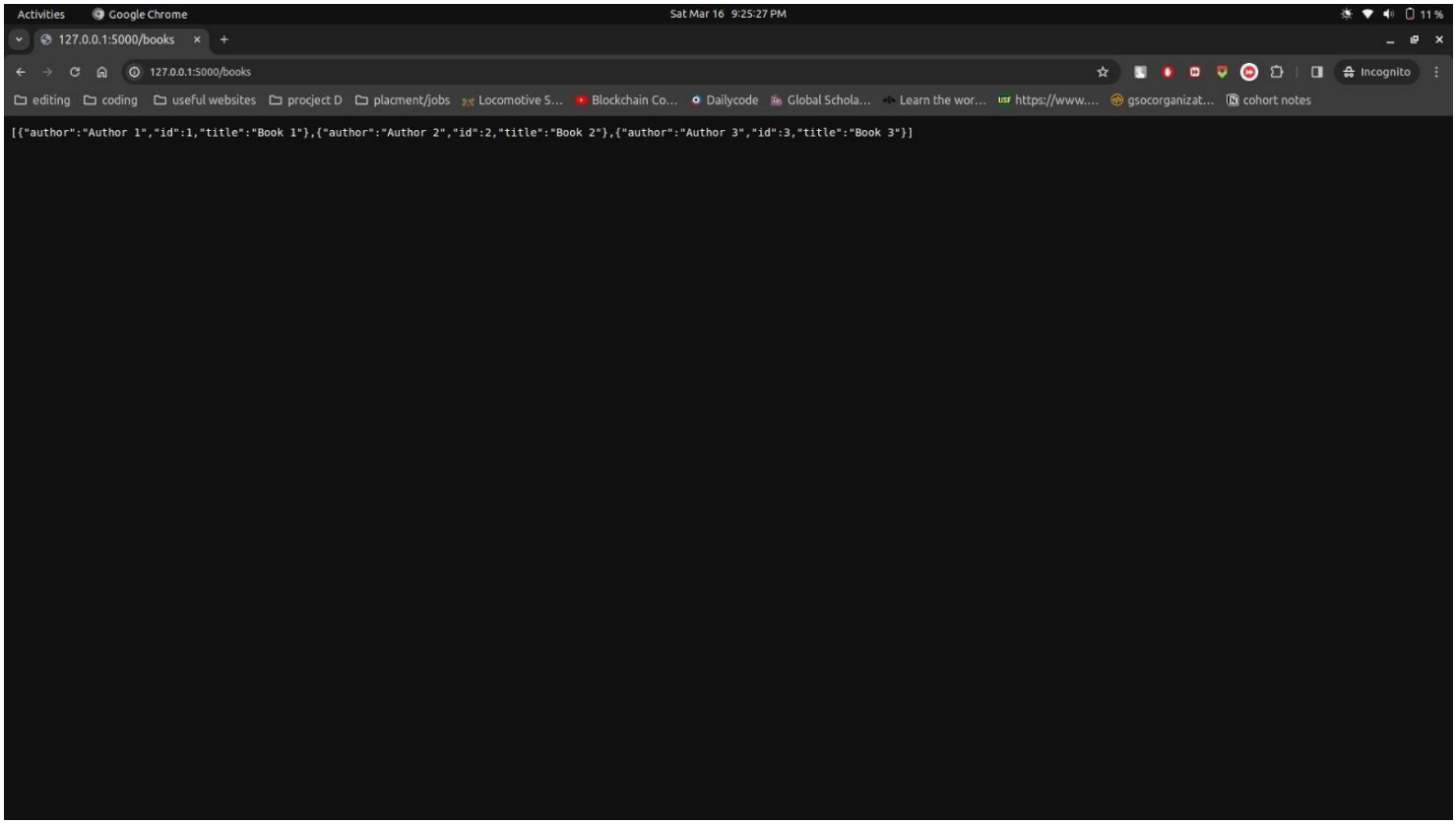
- CODE:

  - 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
11 %
127.0.0.1:5000/books
127.0.0.1:5000/books
Incognito
editing
coding
useful websites
procject 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

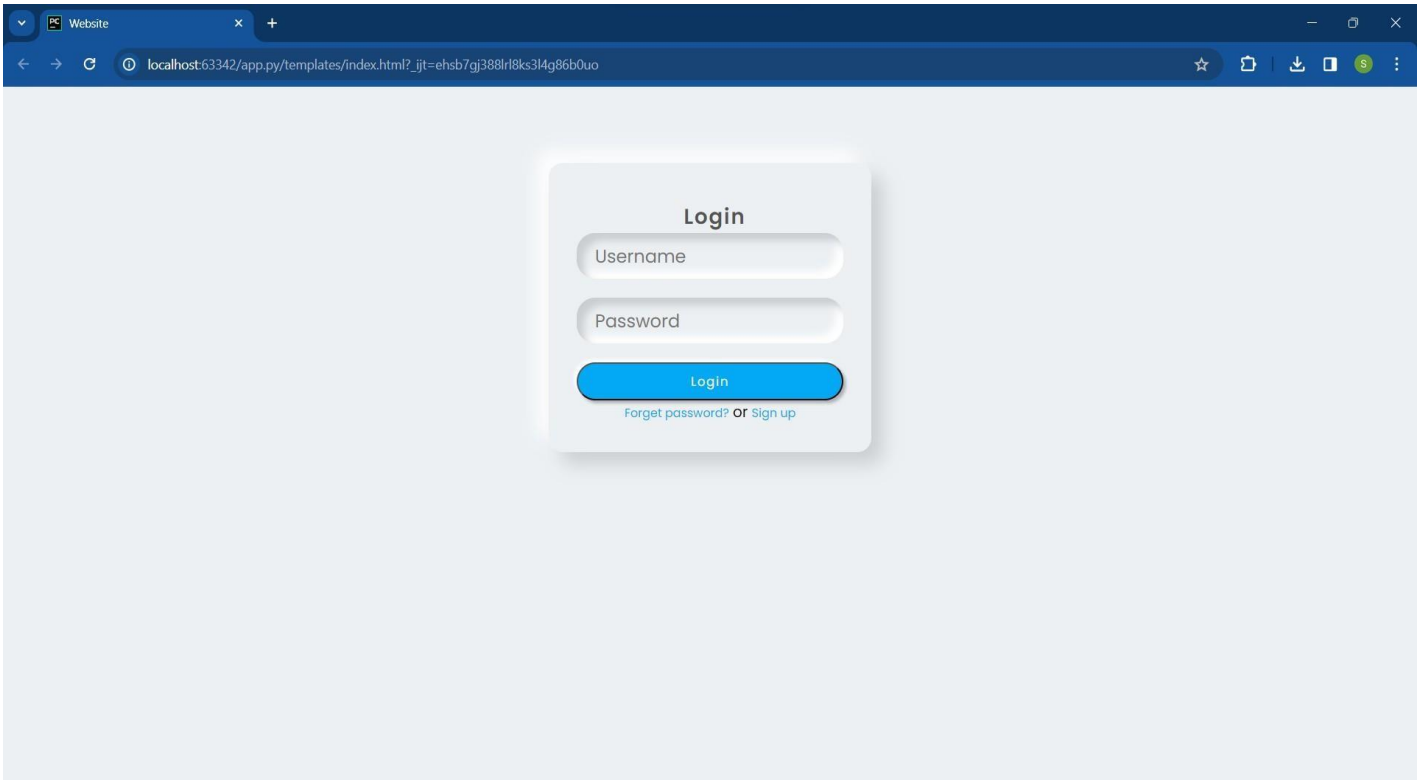
- CODE:

  - 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

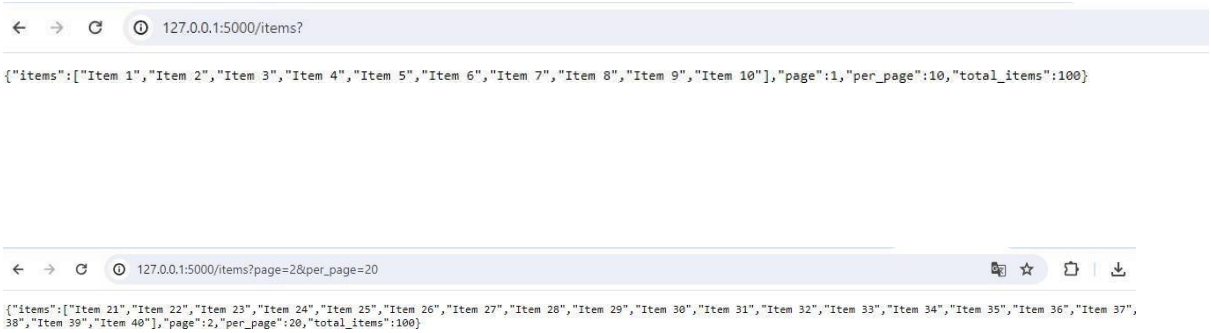
- AIM: A program that creates a RESTful API that paginates the results of a query to improve performance

- 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: