

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**BACHELOR OF TECHNOLOGY**

**Programming in Python with Full Stack Development**

**IV – SEM**

**COMPUTER SCIENCE & ENGINEERING**

**DEPARTMENT**

Laboratory Manual

**CERTIFICATE**

This is to certify that

Mr. /Miss. **RAMCHANDRA** with Enrolment No. **2203031050543** has successfully completed his/her laboratory experiments in the **Programming in Python with Full Stack Development** (303105258) from the Department of **COMPUTER SCIENCE & ENGINEERING** during the academic year **2023 – 2024.**



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

Head of Department: ...........................................

**SET-1**

**1. A program that converts temperatures from Fahrenheit to Celsius and vice versa.**

**Code:**

# Fahrenheit to Celsius

def FtoC(Fahrenheit):

return(Fahrenheit- 32.0) \* 5.0 / 9.0

Fahrenheit = 100

Celsius = FtoC(Fahrenheit)

print (Celsius)

# Celsius to Fahrenheit

def CtoF(Celsius):

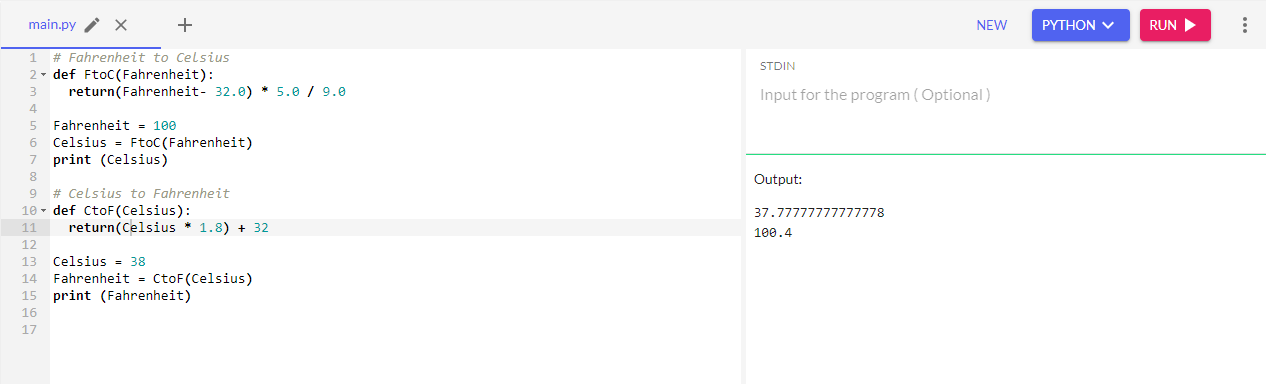
return(celsius \* 1.8) + 32

Celsius = 38

Fahrenheit = CtoF(Celsius)

print (Fahrenheit)

**Output:**

****

**2. A program that calculates the area and perimeter of a rectangle.**

**Code:**

length = int(input())

width = int(input())

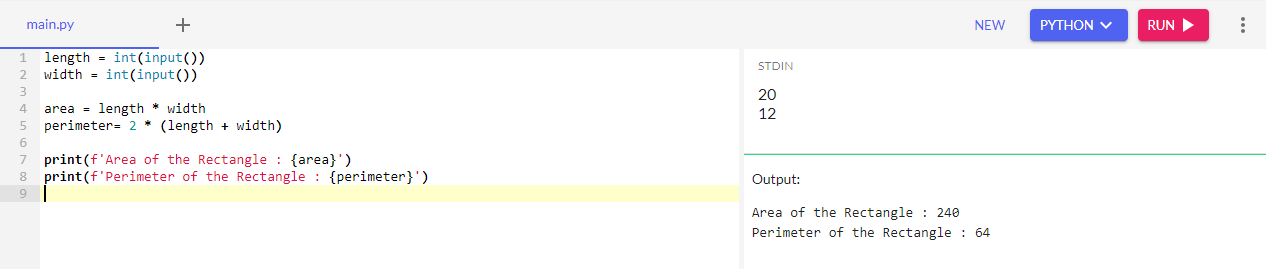
area = length \* width

perimeter= 2 \* (length + width)

print(f'Area of the Rectangle : {area}')

print(f'Perimeter of the Rectangle : {perimeter}')

**Output:**

****

**3. A program that generates a random password of a specified length.**

**Code:**

import string

import secrets

symbols = ['\*', '%', '£'] # Can add more

password = ""

for \_ in range(9):

password += secrets.choice(string.ascii\_lowercase)

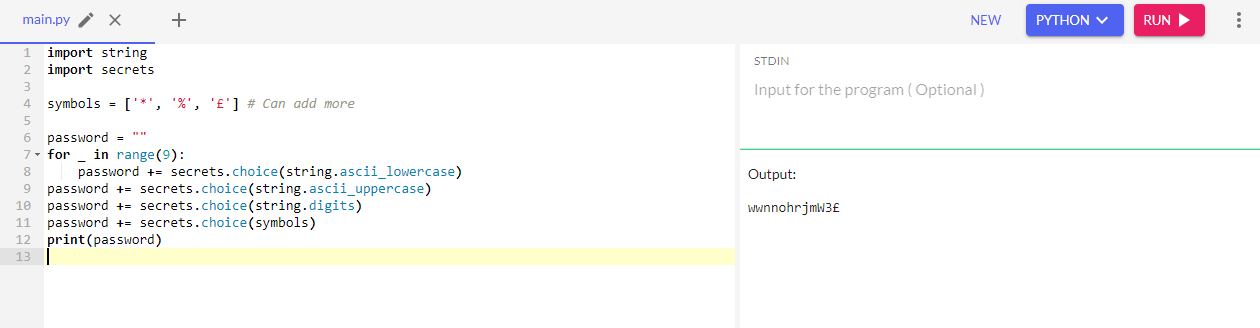
password += secrets.choice(string.ascii\_uppercase)

password += secrets.choice(string.digits)

password += secrets.choice(symbols)

print(password)

**Output:**

****

**4. A program that calculates the average of a list of numbers.**

**Code:**

# Python program to get average of a list

def Average(lst):

return sum(lst) / len(lst)

# Driver Code

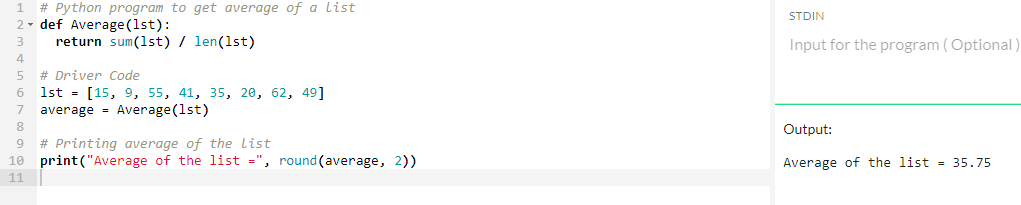
lst = [15, 9, 55, 41, 35, 20, 62, 49]

average = Average(lst)

# Printing average of the list

print("Average of the list =", round(average, 2))

**Output:**

****

**5. A program that checks if a given year is a leap year.**

**Code:**

def ISLP(y):

if((y % 400 == 0)or(y % 100 != 0)and(y % 4 == 0)):

return 1;

else:

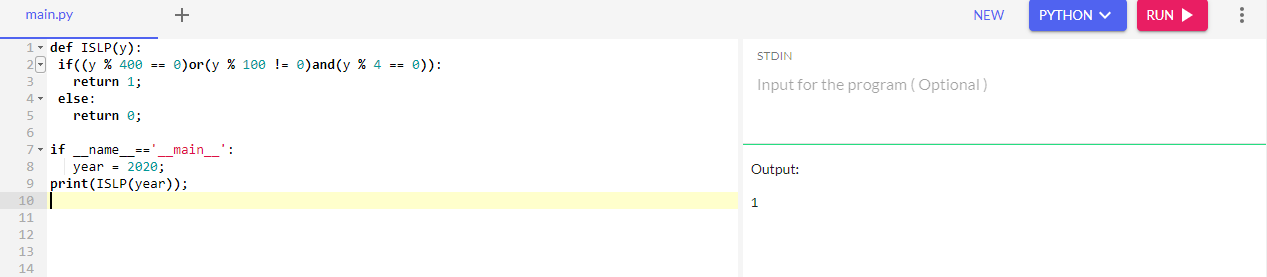
return 0;

if \_\_name\_\_=='\_\_main\_\_':

year = 2020;

print(ISLP(year));

**Output:**

****

**6. A program that calculates the factorial of a number.**

**Code:**

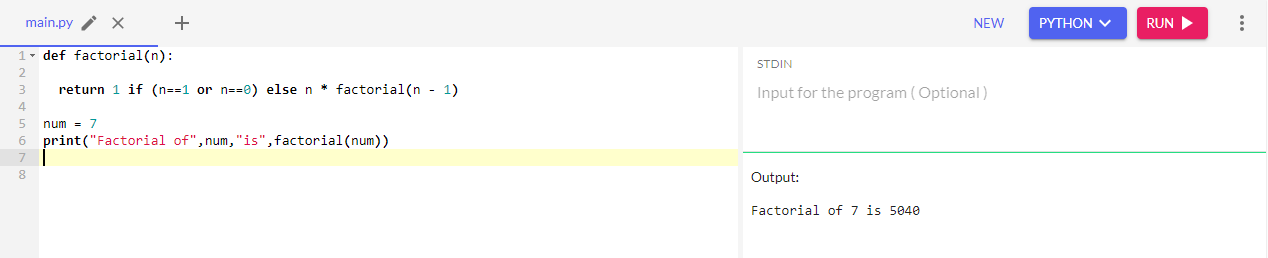
def factorial(n):

return 1 if (n==1 or n==0) else n \* factorial(n - 1)

num = 7

print("Factorial of",num,"is",factorial(num))

**Output:**

****

**7. A program that checks if a given string is a palindrome.**

**Code:**

def isPalindrome(s):

return s == s[::-1]

s = "malayalam"

ans = isPalindrome(s)

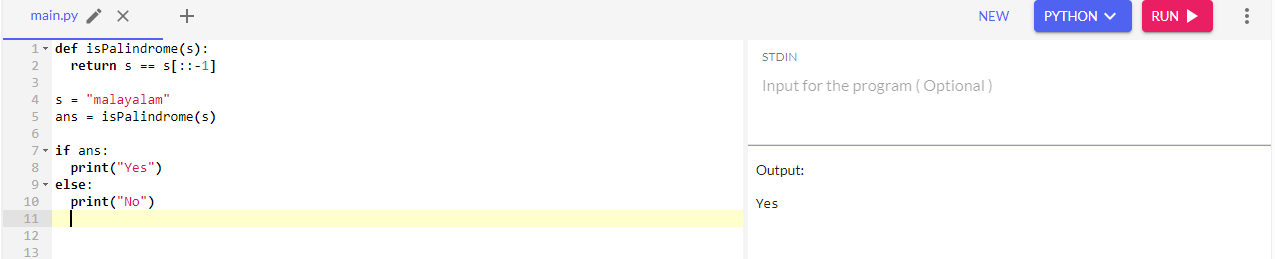
if ans:

print("Yes")

else:

print("No")

**Output:**

****

**8. A program that sorts a list of numbers in ascending or descending order.**

**Code:**

alphabets = ['a','e','d','c','b']

alphabets.sort()

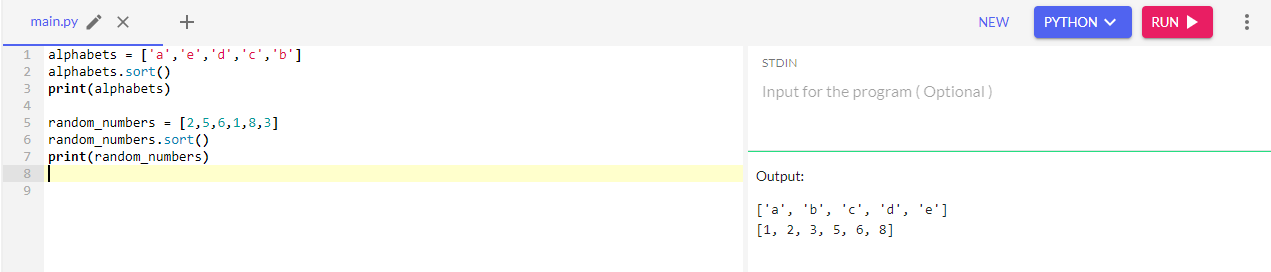
print(alphabets)

random\_numbers = [2,5,6,1,8,3]

random\_numbers.sort()

print(random\_numbers)

**Output:**

****

**9. A program that generates a multiplication table for a given number.**

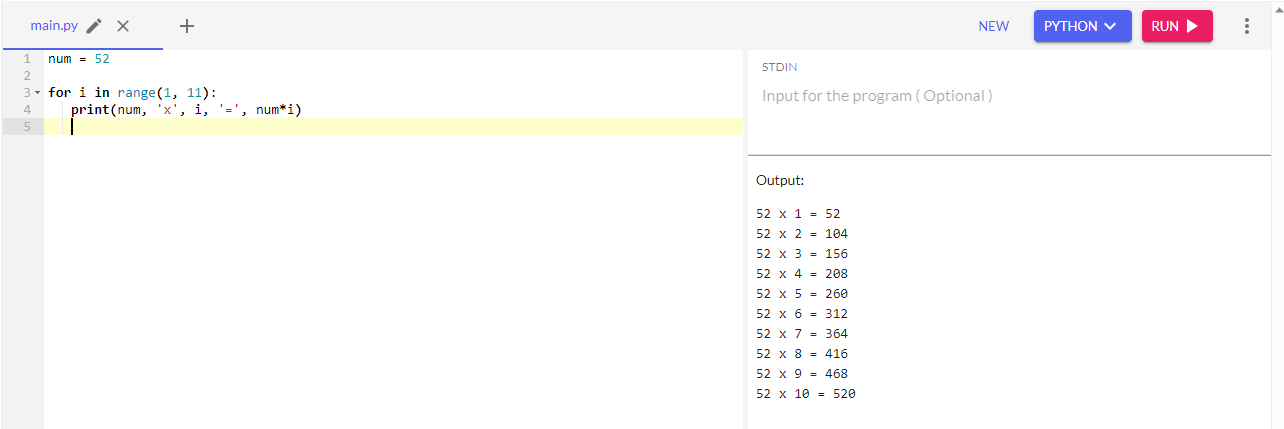
**Code:**

num = 52

for i in range(1, 11):

print(num, 'x', i, '=', num\*i)

**Output:**

****

**10. A program that converts a given number from one base to another.**

**Code:**

def decimal\_to\_binary(dec):

decimal = int(dec)

print(decimal, " in Binary : ", bin(decimal))

def decimal\_to\_octal(dec):

decimal = int(dec)

print(decimal, "in Octal : ", oct(decimal))

def decimal\_to\_hexadecimal(dec):

decimal = int(dec)

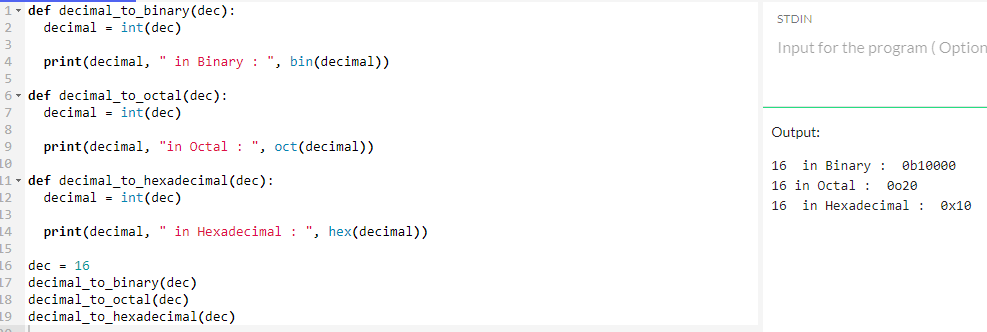
print(decimal, " in Hexadecimal : ", hex(decimal))

dec = 16

decimal\_to\_binary(dec)

decimal\_to\_octal(dec)

decimal\_to\_hexadecimal(dec)

**Output:**

**SET-2**

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

**Program:-**

class BankAccount:

    def \_\_init\_\_(self, account\_number, customer\_name, balance=0.0):

        self.account\_number = account\_number

        self.customer\_name = customer\_name

        self.balance = balance

    def deposit(self, amount):

        self.balance += amount

        print(f"Deposited ₹{amount}. New balance: ₹{self.balance}")

    def withdraw(self, amount):

        if 0 < amount <= self.balance:

            self.balance -= amount

            print(f"Withdrew ₹{amount}. New balance: ₹{self.balance}")

        else:

            print("Invalid withdrawal amount or insufficient funds.")

# Example Usage:

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

    # Create accounts

    account1 = BankAccount(1, "John Doe", balance=10000.0)

    account2 = BankAccount(2, "Jane Smith", balance=5000.0)

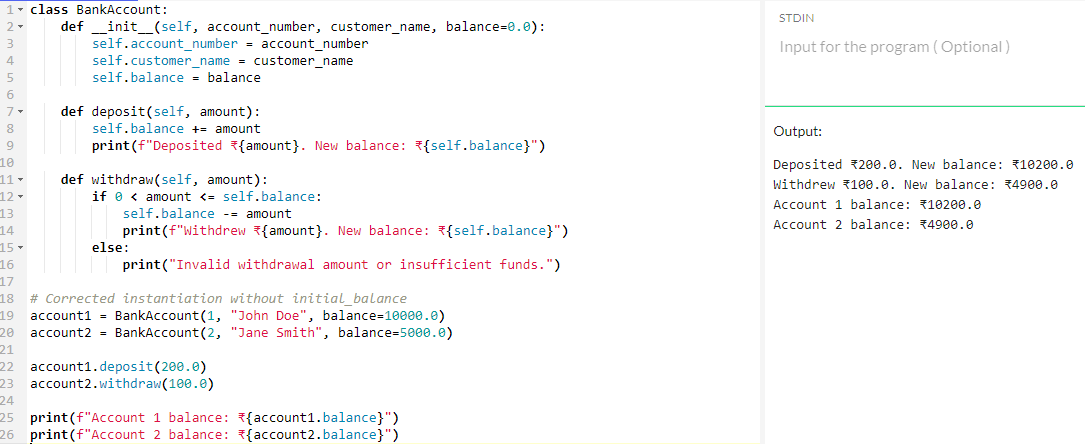
    # Perform transactions

    account1.deposit(200.0)

    account2.withdraw(100.0)

    print(f"Account 1 balance: ₹{account1.balance}")

    print(f"Account 2 balance: ₹{account2.balance}")

**OUTPUT:**

1. **A program that simulates a school management system, with classes for the students, the teachers, and the courses**

**Program:-**

class Person:

    def \_\_init\_\_(self, id, name):

        self.id = id

        self.name = name

class Student(Person):

    def \_\_init\_\_(self, student\_id, student\_name, grade=0):

        super().\_\_init\_\_(student\_id, student\_name)

        self.grade = grade

    def display\_info(self):

        print(f"Student ID: {self.id}, Name: {self.name}, Grade: {self.grade}")

class Teacher(Person):

    def \_\_init\_\_(self, teacher\_id, teacher\_name, subject):

        super().\_\_init\_\_(teacher\_id, teacher\_name)

        self.subject = subject

    def display\_info(self):

        print(f"Teacher ID: {self.id},\nName: {self.name},\nSubject: {self.subject}")

class Course:

    def \_\_init\_\_(self, course\_code, course\_name, teacher, students=None):

        self.course\_code = course\_code

        self.course\_name = course\_name

        self.teacher = teacher

        self.students = students if students else []

    def add\_student(self, student):

        self.students.append(student)

    def display\_info(self):

        print(f"Course Code: {self.course\_code},\nCourse Name: {self.course\_name}")

        self.teacher.display\_info()

        print("Students:")

        for student in self.students:

            student.display\_info()

        print()

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

    # Create students, teachers, and courses

    student1 = Student(1, "John Doe", grade=10)

    student2 = Student(2, "Jane Smith", grade=11)

    teacher1 = Teacher(101, "Mr. Johnson", subject="Math")

    teacher2 = Teacher(102, "Ms. Davis", subject="Science")

    math\_course = Course("MATH101", "Algebra", teacher1, students=[student1])

    science\_course = Course("SCI201", "Biology", teacher2, students=[student2])

    # Add students to courses and display information

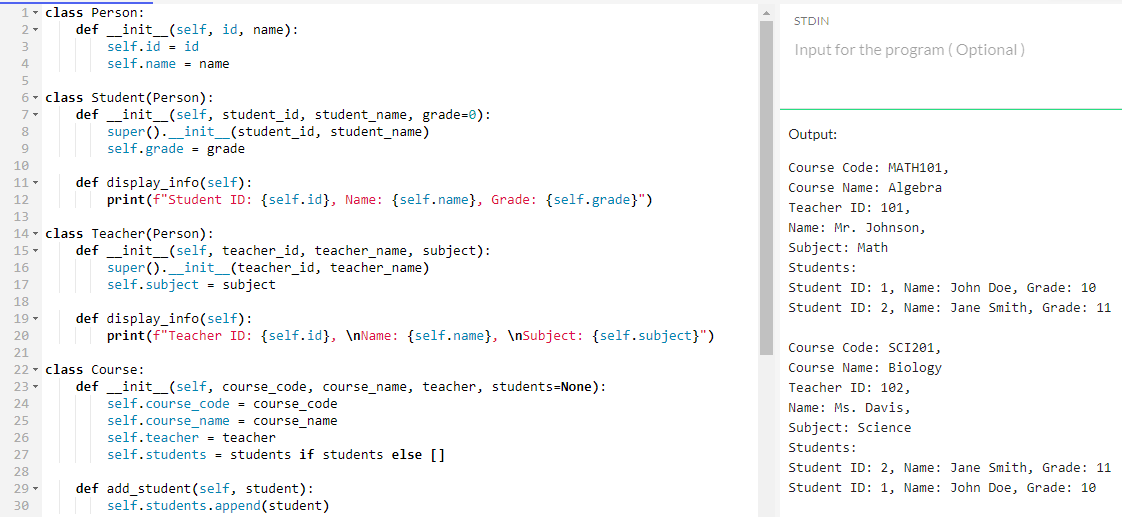
    math\_course.add\_student(student2)

    science\_course.add\_student(student1)

    math\_course.display\_info()

    science\_course.display\_info()

**OUTPUT:**

****

1. **A program that reads a text file and counts the number of words in it.**

**Program:-**

def count\_words(file\_path):

    try:

        with open(file\_path, 'r') as file:

            content = file.read()

            word\_count = len(content.split())

            return word\_count

    except FileNotFoundError:

        print(f"Error: File '{file\_path}' not found.")

        return None

    except Exception as e:

        print(f"An error occurred: {e}")

        return None

# Example Usage:

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

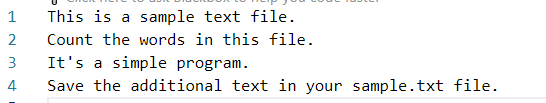
    file\_path = "sample.txt"  # Replace with the path to your text file

    word\_count = count\_words(file\_path)

    if word\_count is not None:

        print(f"The number of words in the file is: {word\_count}")

**Attach file Name: Sample.txt**

****

**OUTPUT:**

****

1. **A program that reads a CSV file and calculates the average of the values in a specified column.**

**Program:-**

import csv

def calculate\_column\_average(file\_path, column\_index):

    try:

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

            csv\_reader = csv.reader(csv\_file)

            data = list(csv\_reader)

            if not data:

                print(f"Error: File '{file\_path}' is empty.")

                return None

            column\_values = [float(row[column\_index]) for row in data[1:]]

            average = sum(column\_values) / len(column\_values)

            return average

    except FileNotFoundError:

        print(f"Error: File '{file\_path}' not found.")

        return None

    except IndexError:

        print(f"Error: Specified column index is out of bounds.")

        return None

    except Exception as e:

        print(f"An error occurred: {e}")

        return None

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

    file\_path = "data.csv” # Replace with the path to your CSV file

    column\_index = 1 # Replace with the index of the column you want to calculate the average for (0-based)

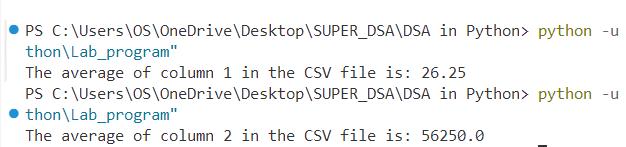
    average = calculate\_column\_average(file\_path, column\_index)

    if average is not None:

        print(f"The average of column {column\_index} in the CSV file is: {average}")

**Attach file: - data.csv**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.No. | Name | Age | Salary |
| 1. | John | 25 | 50000 |
| 2. | Jane | 30 | 60000 |
| 3. | Bob | 22 | 45000 |
| 4. | Alice | 28 | 70000 |

**OUTPUT:**

1. **A program that reads an Excel file and prints the data in a tabular format**

**Program:-**

import pandas as pd

def write\_data\_to\_excel(output\_file\_path):

    try:

        # Create a DataFrame with sample data

        data = {

            'Name': ['John', 'Alice', 'Bob'],

            'Age': [25, 30, 22],

            'City': ['New York', 'San Francisco', 'Seattle']

        }

        df = pd.DataFrame(data)

        # Print the DataFrame

        print("Data to be written:")

        print(df)

        # Write the DataFrame to an Excel file

        df.to\_excel(output\_file\_path, index=False)

        print(f"\nData written to '{output\_file\_path}' successfully.")

    except Exception as e:

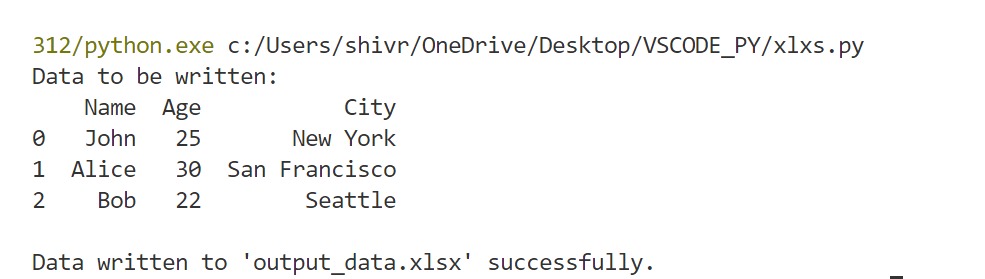
        print(f"An error occurred: {e}")

# Replace 'output\_data.xlsx' with the desired output file path

output\_excel\_file\_path = 'output\_data.xlsx'

write\_data\_to\_excel(output\_excel\_file\_path)

**OUTPUT:**

****

**SET-3**

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

# --->pip install Flask

# ---> python app.py

**your\_project\_folder/**

**│**

**├── app.py**

**└── Templates/**

**└── index.html**

**View the Page:** Open your web browser and navigate to **http://127.0.0.1:5000/** to view your served HTML page

**Program:**

**File: app.py**

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

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

    return render\_template('index.html')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**File: index.html**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Simple Web Server</title>

</head>

<body>

    <h1>Welcome to my Simple Web Server</h1>

    <p>This is a static HTML page served by a Flask web server.</p>

</body>

</html>

**Just click the link:**

\* Running on http://127.0.0.I:5000

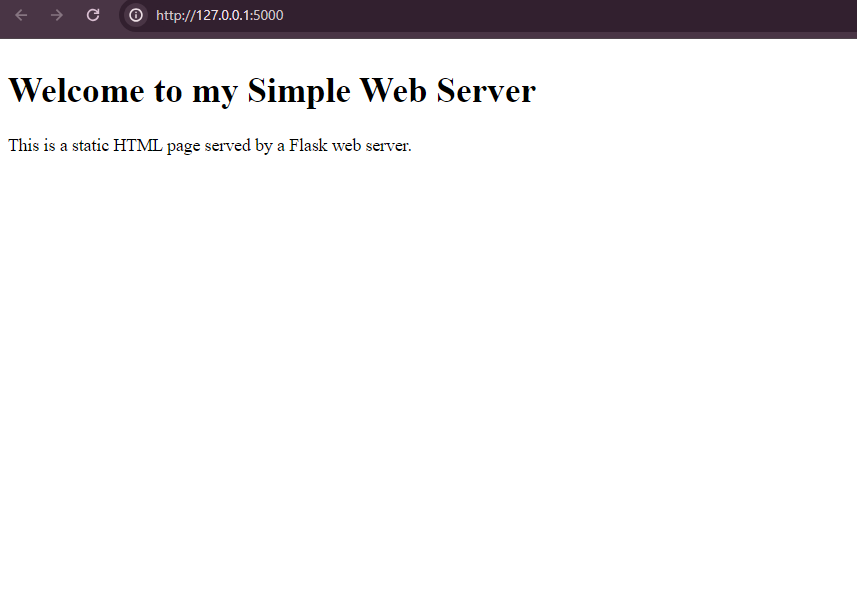
-------------> Press CTRL+C to quit

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 866-921-880

**OUTPUT:**

****

1. **A program that creates a web application that allows users to register and login.**

**your\_project\_folder/**

**│**

**├── app.py**

**└── Templates/**

**└── index.html**

**└── login.html**

**└── register.html**

**Program:**

**index.html:-**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Home</title>

    <style>

        #home{

            position: absolute;

            top: 28%;

            left: 28%;

            height: 200px;

            width: 600px;

            display: block;

            justify-content: space-evenly;

            align-items: center;

            background-color: rgb(244, 241, 237);

            text-align: center;

            border: 2px solid black;

            border-radius: 4px;

        }

    </style>

</head>

<body>

   <div id="home">

     <h1>Welcome to our Web App</h1>

   <h3> <a href="{{ url\_for('login') }}">Login</a></h3>

    <h3><a href="{{ url\_for('register') }}">Register</a></h3>

   </div>

</body>

</html>

**login.html:-**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Login</title>

    <style>

        .login {

            position: absolute;

            top: 28%;

            left: 28%;

            height: 250px;

            width: 600px;

            display: block;

            justify-content: space-evenly;

            align-items: center;

            background-color: rgb(244, 241, 237);

            text-align: center;

            border: 2px solid black;

            border-radius: 4px;

        }

        .login input {

            padding: 8px;

            width: 80%;

            border: 1px solid #ccc;

            border-radius: 4px;

        }

        .login button {

            padding: 8px 16px;

            background-color: #007bff;

            color: white;

            border: none;

            border-radius: 4px;

            cursor: pointer;

        }

        .login button:hover {

            background-color: #0056b3;

        }

    </style>

</head>

<body>

    <div class="login">

        <h1>Login</h1>

        <form method="POST" action="" id="form1">

            <div>

                <label for="email">Email:</label>

                <input type="email" id="email" name="email" size="32" required>

            </div>

            <br>

            <div>

                <label for="password">Password:</label>

                <input type="password" id="password" name="password" size="32" required>

            </div>

            <br>

            <div>

                <button type="submit">Login</button>

            </div>

        </form>

    </div>

</body>

</html>

**register.html:-**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Register</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: white;

            margin: 0;

            padding: 0;

        }

        .container {

            width: 80%;

            max-width: 600px;

            margin: 50px auto;

           background-color: rgb(244, 241, 237);

            padding: 20px;

            border-radius: 8px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

            border: 2px solid black;

        }

        h1 {

            text-align: center;

            margin-bottom: 20px;

        }

        form {

            display: flex;

            flex-direction: column;

        }

        label {

            margin-bottom: 5px;

            font-weight: bold;

        }

        input {

            padding: 10px;

            margin-bottom: 15px;

            border: 1px solid #ccc;

            border-radius: 5px;

        }

        input[type="submit"] {

            background-color: #007bff;

            color: #fff;

            border: none;

            border-radius: 5px;

            cursor: pointer;

        }

        input[type="submit"]:hover {

            background-color: #0056b3;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Register</h1>

        <form method="POST" action="">

            {{ form.hidden\_tag() }}

            <div>

                <label for="username">Username:</label>

                {{ form.username(size=32) }}

            </div>

            <div>

                <label for="email">Email:</label>

                {{ form.email(size=32) }}

            </div>

            <div>

                <label for="password">Password:</label>

                {{ form.password(size=32) }}

            </div>

            <div>

                <label for="confirm\_password">Confirm Password:</label>

                {{ form.confirm\_password(size=32) }}

            </div>

            <div>

                <input type="submit" value="Register">

            </div>

        </form>

    </div>

</body>

</html>

**Program: App.py:-**

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

from flask import Flask, render\_template, redirect, url\_for, flash

from flask\_wtf import FlaskForm

from wtforms import StringField, PasswordField, SubmitField

from wtforms.validators import DataRequired, Email, EqualTo

from flask\_bcrypt import Bcrypt

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'your\_secret\_key\_here'

bcrypt = Bcrypt(app)

users = []

class RegistrationForm(FlaskForm):

    username = StringField('Username', validators=[DataRequired()])

    email = StringField('Email', validators=[DataRequired(), Email()])

    password = PasswordField('Password', validators=[DataRequired()])

    confirm\_password = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])

    submit = SubmitField('Sign Up')

class LoginForm(FlaskForm):

    email = StringField('Email', validators=[DataRequired(), Email()])

    password = PasswordField('Password', validators=[DataRequired()])

    submit = SubmitField('Log In')

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/register', methods=['GET', 'POST'])

def register():

    form = RegistrationForm()

    if form.validate\_on\_submit():

        hashed\_password = bcrypt.generate\_password\_hash(form.password.data).decode('utf-8')

        user = {'username': form.username.data, 'email': form.email.data, 'password': hashed\_password}

        users.append(user)

        flash('Account created successfully! Please log in.', 'success')

        return redirect(url\_for('login'))

    return render\_template('register.html', form=form)

@app.route('/login', methods=['GET', 'POST'])

def login():

    form = LoginForm()

    if form.validate\_on\_submit():

        for user in users:

            if user['email'] == form.email.data:

                if bcrypt.check\_password\_hash(user['password'], form.password.data):

flash('Login successful!', 'success')

                    return redirect(url\_for('index'))

        flash('Invalid email or password. Please try again.', 'danger')

    return render\_template('login.html', form=form)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Just click the link:**

\* Running on http://127.0.0.1:5000

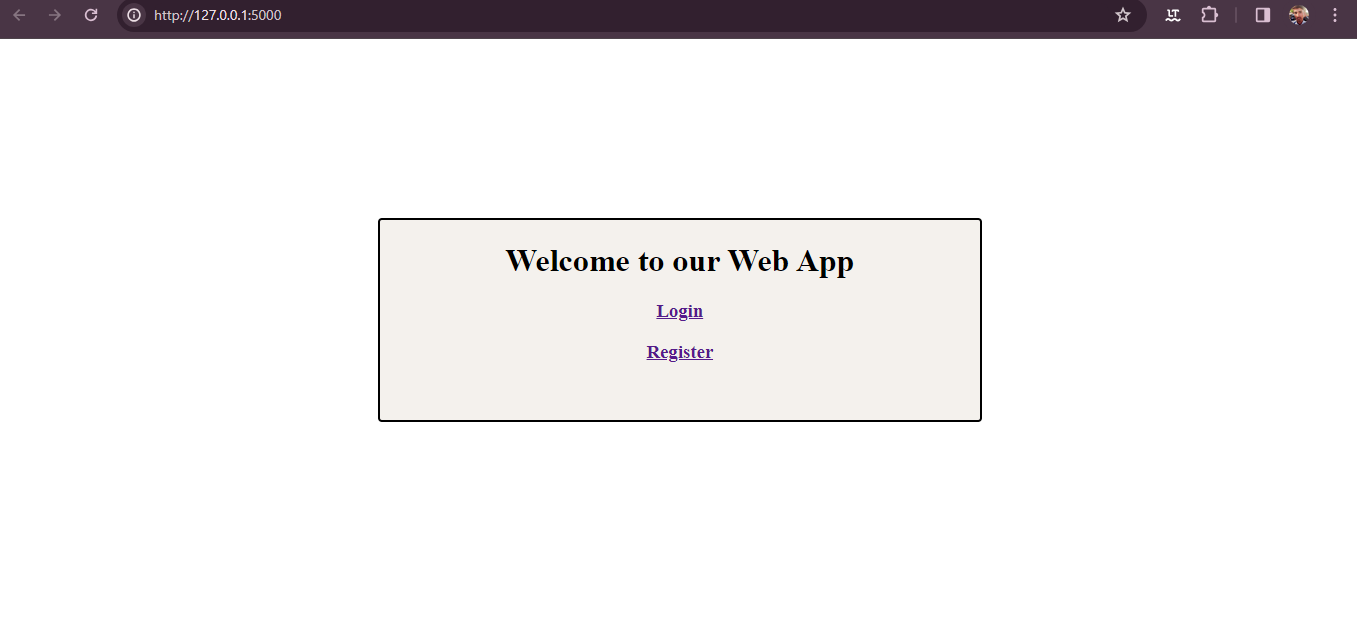
------------> Press CTRL+C to quit

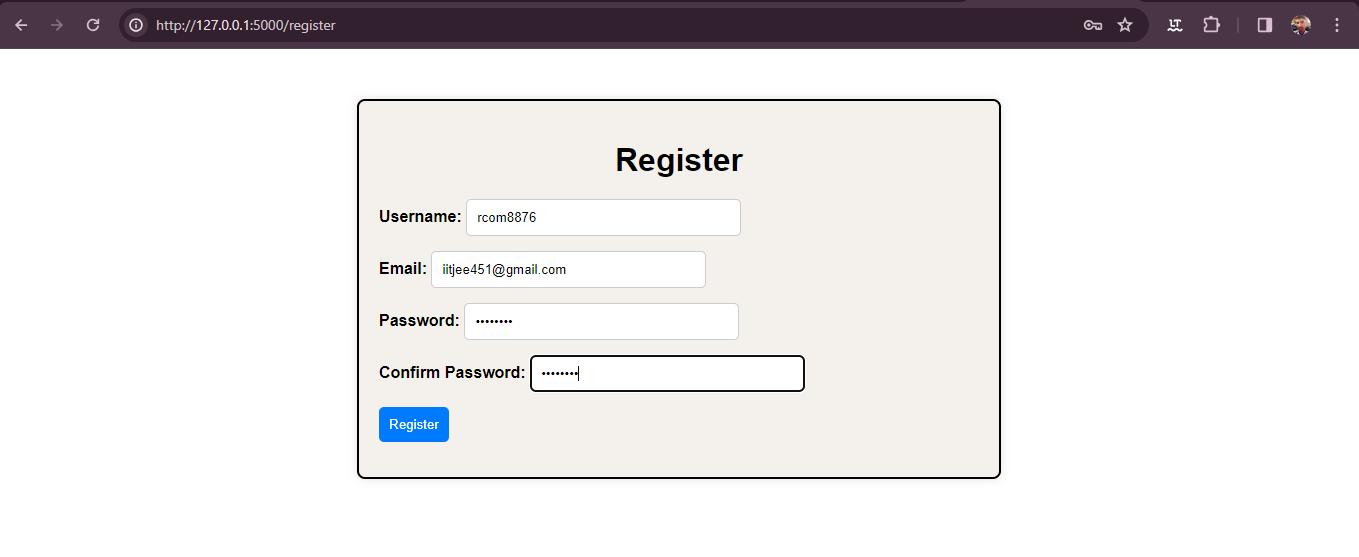
\* Restarting with stat

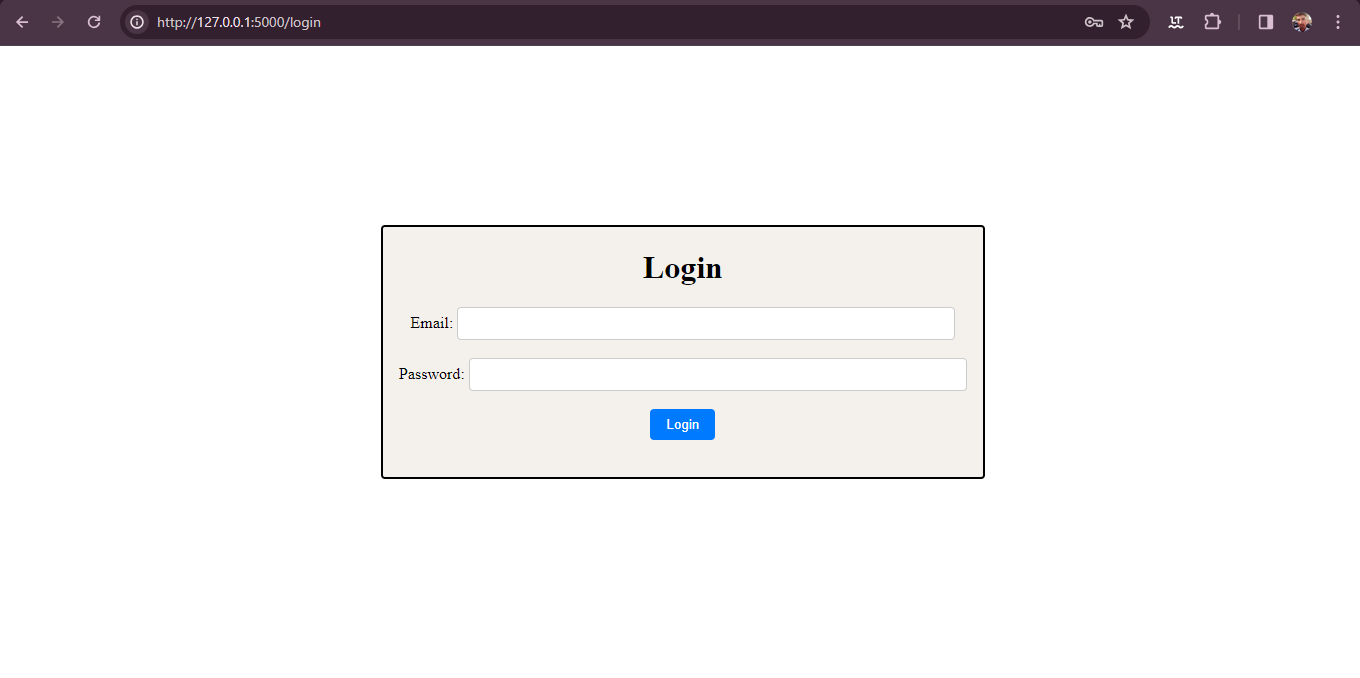
\* Debugger is active!

\* Debugger PIN: 845-411-330

**OUTPUT:**

****

**OUTPUT:**

**OUTPUT:**

1. **A program that creates a web application that allows users to upload and download files.**

**your\_project\_folder/**

**│**

**├── app.py**

**└── Templates/**

**└── index.html**

**Program: Index.html:**

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

<!DOCTYPE html>

<html>

<head>

    <title>File Upload and Download</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f4f4f4;

            margin: 0;

            padding: 0;

        }

        .container {

            max-width: 800px;

            margin: 20px auto;

            padding: 20px;

            background-color: #fff;

            border-radius: 8px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

            border: 2px solid black;

        }

        h1 {

            margin-bottom: 20px;

        }

        form input[type="file"] {

            margin-bottom: 10px;

        }

        ul {

            list-style-type: none;

            padding: 0;

        }

        ul li {

            margin-bottom: 5px;

        }

        ul li a {

            text-decoration: none;

            color: #007bff;

            display: inline-block;

            padding: 5px 10px;

            border: 1px solid #007bff;

            border-radius: 4px;

        }

        ul li a:hover {

            background-color: #007bff;

            color: #fff;

            border-color: #007bff;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>File Upload</h1>

        <form action="/upload" method="post" enctype="multipart/form-data">

            <input type="file" name="file">

            <input type="submit" value="Upload">

        </form>

        <h1>File Download</h1>

        <ul>

            {% for filename in filenames %}

                <li><a href="/download/{{ filename }}">{{ filename }}</a></li>

            {% endfor %}

        </ul>

    </div>

</body>

</html>

**App.py:**

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

from flask import Flask, render\_template, request, send\_file

import os

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = 'uploads'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

@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 'File uploaded successfully'

@app.route('/download/<filename>')

def download\_file(filename):

    return send\_file(os.path.join(app.config['UPLOAD\_FOLDER'], filename), as\_attachment=True)

if \_\_name\_\_ == '\_\_main\_\_':

    if not os.path.exists(UPLOAD\_FOLDER):

        os.makedirs(UPLOAD\_FOLDER)

    app.run(debug=True)

**Just click the link:**

\* Running on http://127.0.0.1:5000

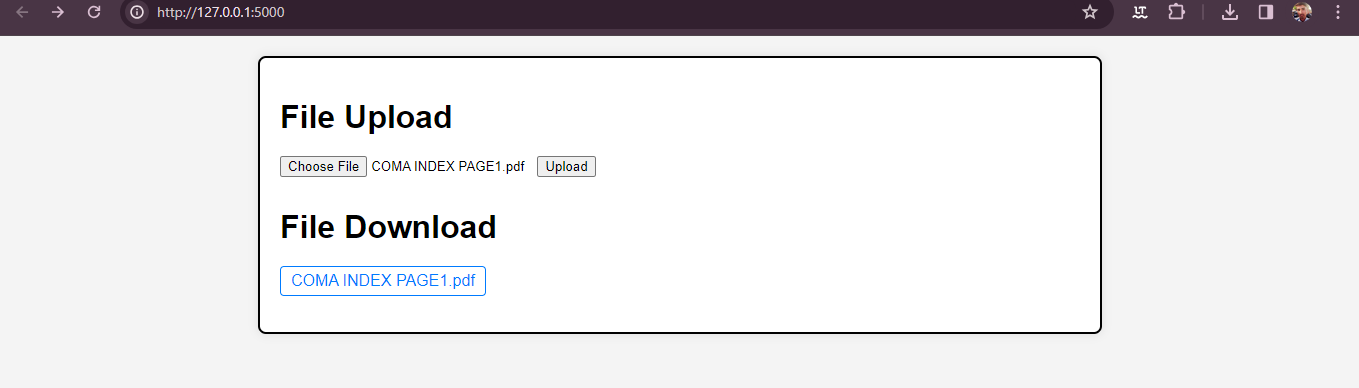
---------------> Press CTRL+C to quit

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 845-411-330

**OUTPUT:**

****

1. **A program that creates a web application that displays data from a database in a tabular format.**

**your\_project\_folder/**

**│**

**├── app.py**

**└── Templates/**

**└── index.html**

**Program: 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

db = SQLAlchemy(app)

# Define the 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)

    email = db.Column(db.String(100), nullable=False, unique=True)  # Ensure email is unique

# Sample data for demonstration

sample\_data = [

    {'name': 'John', 'age': 25, 'email': 'john@example.com'},

    {'name': 'Alice', 'age': 30, 'email': 'alice@example.com'},

    {'name': 'Bob', 'age': 22, 'email': 'bob@example.com'}

]

# Populate the database with sample data

with app.app\_context():

    db.create\_all()

    for entry in sample\_data:

        # Check if record with same email exists before adding

        if not Person.query.filter\_by(email=entry['email']).first():

            person = Person(name=entry['name'], age=entry['age'], email=entry['email'])

            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, person.email) for person in data], columns=['Name', 'Age', 'Email'])

    # 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)

**Just click the link:**

\* Running on http://127.0.0.1:5000

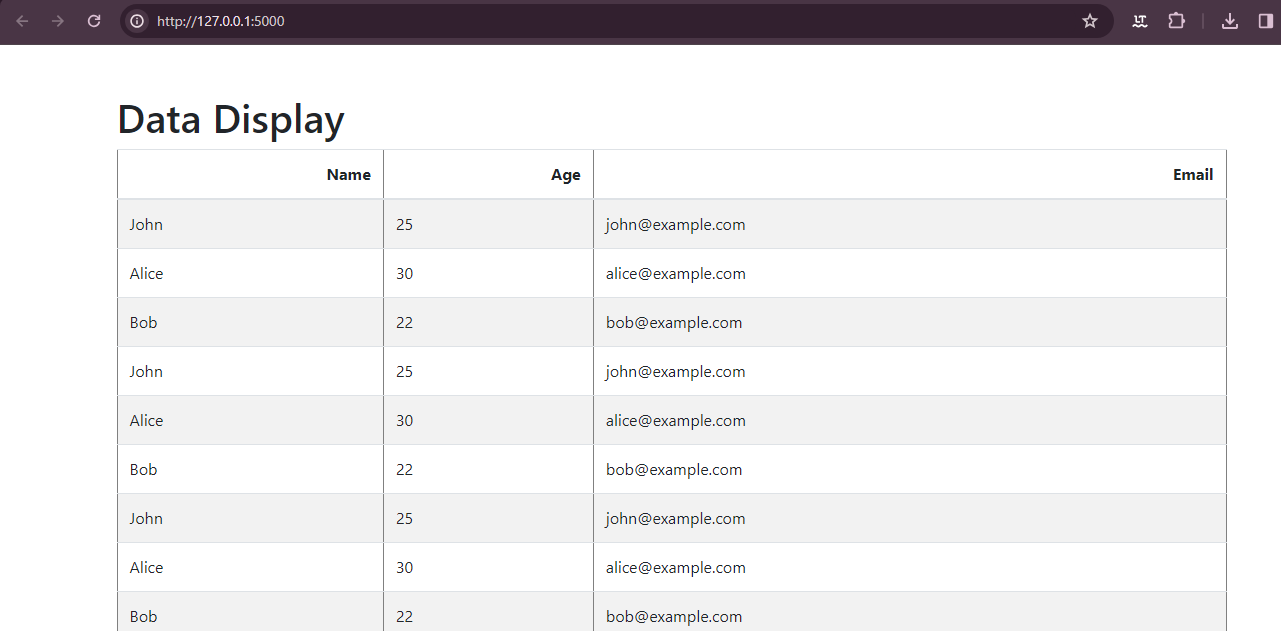
---------------> Press CTRL+C to quit

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 845-411-330

**OUTPUT:**

****

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

**your\_project\_folder/**

**│**

**├── app.py**

**└── Templates/**

**└── index.html**

**Program: 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 Form</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f2f2f2;

            margin: 0;

            padding: 0;

        }

        .container {

            max-width: 500px;

            margin: 50px auto;

            padding: 20px;

            background-color: #fff;

            border-radius: 8px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

            border: 2px solid black;

        }

        h1 {

            text-align: center;

        }

        form {

            text-align: center;

        }

        label {

            display: block;

            margin-bottom: 10px;

        }

        input[type="text"] {

            width: 100%;

            padding: 10px;

            margin-bottom: 20px;

            border: 1px solid #ccc;

            border-radius: 4px;

            box-sizing: border-box;

        }

        button {

            padding: 10px 20px;

            background-color: #007bff;

            color: #fff;

            border: none;

            border-radius: 4px;

            cursor: pointer;

        }

        button:hover {

            background-color: #0056b3;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>User Input Form</h1>

        <form method="post">

            <label for="input\_data">Enter your data:</label><br>

            <input type="text" id="input\_data" name="input\_data" required><br><br>

            <button type="submit">Submit</button>

        </form>

    </div>

</body>

</html>

**App.py:**

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

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route('/', methods=['GET', 'POST'])

def index():

    if request.method == 'POST':

        # Get the data from the form

        input\_data = request.form.get('input\_data')

        # Process the input data (e.g., perform some computation, store in database, etc.)

        processed\_data = process\_input(input\_data)

        # Return the processed data to display on the webpage

        return render\_template('result.html', input\_data=input\_data, processed\_data=processed\_data)

    # Render the HTML template with the form

    return render\_template('index.html')

def process\_input(input\_data):

    # Here, you can perform any processing or manipulation of the input data

    # For demonstration purposes, simply capitalize the input data

    processed\_data = input\_data.upper()

    return processed\_data

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Just click the link:**

\* Running on http://127.0.0.1:5000

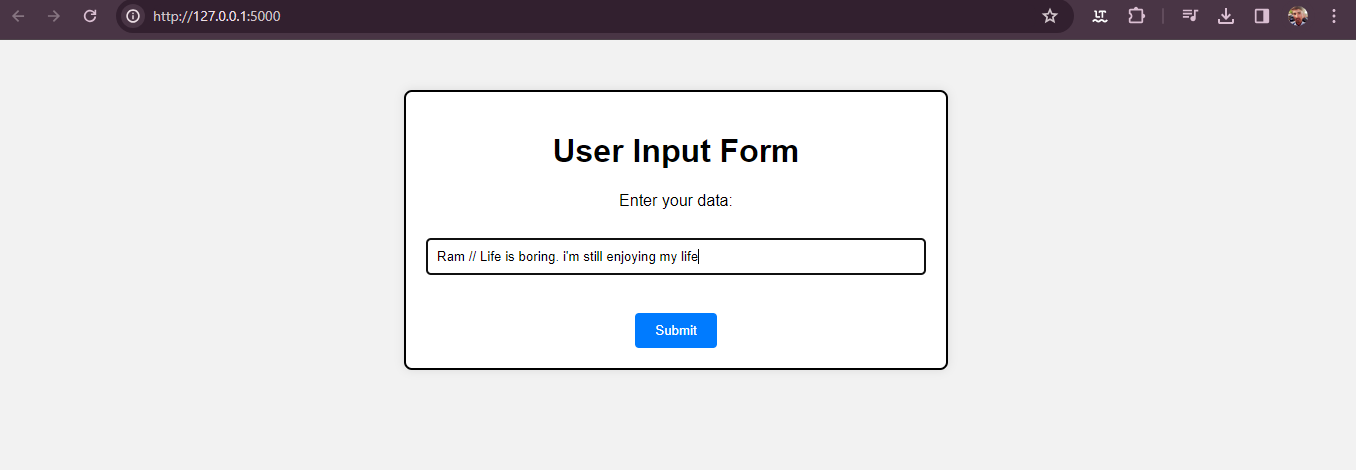
---------------> Press CTRL+C to quit

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 845-411-330

**OUTPUT:**

****

**SET-4**

1. **A program that creates a web application that uses a template engine to generate dynamic HTML pages.**

* First we can create a Virtual Environment for Our Project :

1. **Install virtualenv:**

🡪 pip install virtualenv

1. **Create a virtual environment:**

**🡪 py -m** virtualenv venv

1. **Activate the virtual environment:**
   * 1. **On Windows:**

**🡪** venv\Scripts\activate

* Ensure you have Django installed. You can install it via pip:
  + - 1. **pip install Django**
      2. **Create a Django project:**
         1. django-admin startproject myproject
      3. **Create a Django app:**
         1. cd myproject
         2. python manage.py startapp myapp
* **Define URL patterns:**

In **myproject/urls.py,** define URL patterns to route requests to views:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('myapp.urls')),

]

* **Create views:**

In **myapp/views.py**, define view functions to handle requests and render templates:

from django.shortcuts import render

def index(request):

    context = {

        'message': 'Hello, world!'

    }

    return render(request, 'index.html', context)

* **Create templates:**

Create a templates directory within the myapp directory, and inside it, create an index.html template:

<!-- templates/index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Dynamic HTML Page</title>

</head>

<body>

    <h1>{{ message }}</h1>

</body>

</html>

**Define URL patterns for the app:**

**In myapp/urls.py, define URL patterns to map requests to views:**

from django.urls import path

from . import views

urlpatterns = [

    path('', views.index, name='index'),

]

**Configure settings:**

**In settings.py, add your app to INSTALLED\_APPS:**

INSTALLED\_APPS = [

    ...

    'myapp',

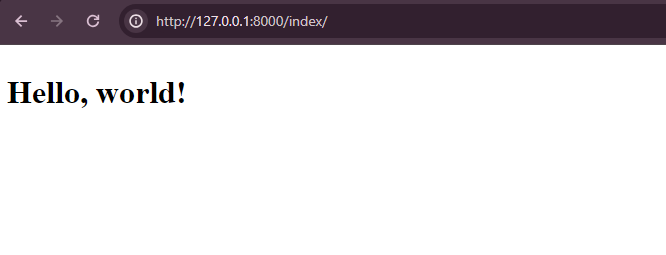
]

**Run the server:**

**Finally, run the development server:**

python manage.py runserver

**OUTPUT:**

****

1. **A program that creates a web application that supports AJAX requests and updates the page without reloading.**

* **Set up Django project:**

🡪Follow the steps mentioned earlier to create a Django project and an app.

* **Create Django views for handling AJAX requests:**

🡪 view.py

# views.py

from django.http import JsonResponse

def update\_content(request):

    data = {

        'message': 'Content updated via AJAX!'

    }

    return JsonResponse(data)

* **Define URL patterns:**

In **myproject/urls.py,** include URL patterns for your app:.

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('myapp.urls')),

]

**In myapp/urls.py, define a URL pattern for this view:**

from django.urls import path

from . import views

urlpatterns = [

    path('update/', views.update\_content, name='update\_content'),

]

* **Create a template with JavaScript:**

Create a template ajax\_template.html in the ajaxapp/templates directory. This template will include JavaScript to make AJAX requests and update the page content:

<!-- index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>AJAX Example</title>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

</head>

<body>

    <div id="content">

        <!-- Content will be updated dynamically -->

    </div>

    <button id="updateButton">Update Content</button>

    <script>

        $(document).ready(function() {

            $('#updateButton').click(function() {

                $.ajax({

                    url: '/update/',

                    type: 'GET',

                    success: function(response) {

                        $('#content').html(response.message);

                    },

                    error: function(xhr, status, error) {

                        console.error('Error:', error);

                    }

                });

            });

        });

    </script>

</body>

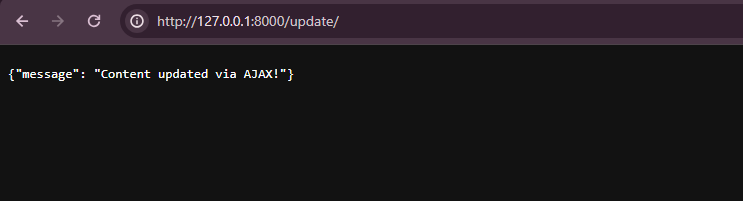
</html>

**Run the Django server:**

**Start the Django development server:**

**🡪 python manage.py runserver**

**OUTPUT:**



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

* **Setting Up a Django Project:**

First, ensure you have Django installed. If not, run:

pip install django

* **Create a new Django project:**

django-admin startproject myproject

* **Create a Django App:**

Inside your project folder, create a new app:

cd myproject

python manage.py startapp myapp

* **Configure Debugging:**

In your settings.py, set **DEBUG = True** (this is essential for debugging).

Add **'myapp'**, to your INSTALLED\_APPS.

* **Create a View:**

In **myapp/views.py**, create a simple view:

from django.shortcuts import render

def hello\_world(request):

    # Simulate an error for debugging

    my\_variable = 1 / 0

    return render(request, 'hello.html')

* **Create a Template:**

Create a template file named **hello.html** in **myapp/templates**

<!DOCTYPE html>

<html>

  <head>

    <title>Hello World</title>

  </head>

  <body>

    <h1>

      Hello, World!

      <p>

        Django's built-in debugging features to troubleshoot errors and

        exceptions

      </p>

    </h1>

  </body>

</html>

* **URL Configuration:**

In **myproject/urls.py**, add a URL pattern for your view

from django.urls import path

from myapp.views import hello\_world

urlpatterns = [

    path('hello/', hello\_world, name='hello'),

]

**OUTPUT:**

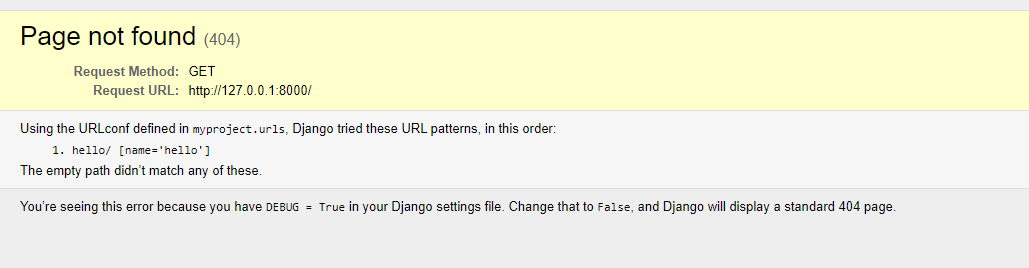
* Visit **http://127.0.0.1:8000/hello/** in your browser.

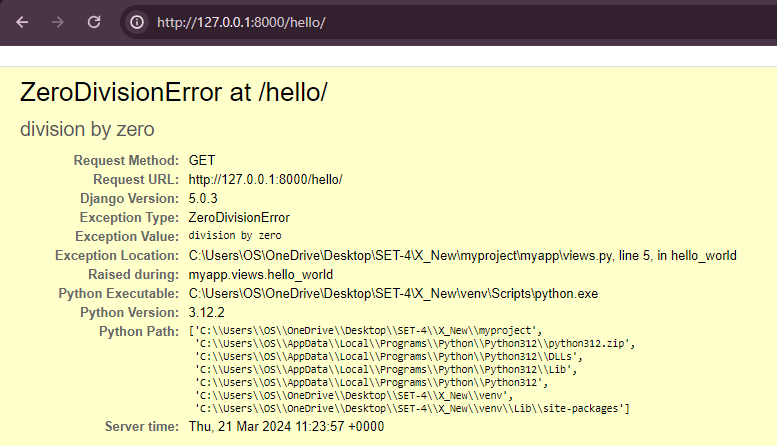
You’ll encounter a ZeroDivisionError due to the line my\_variable = 1 / 0.

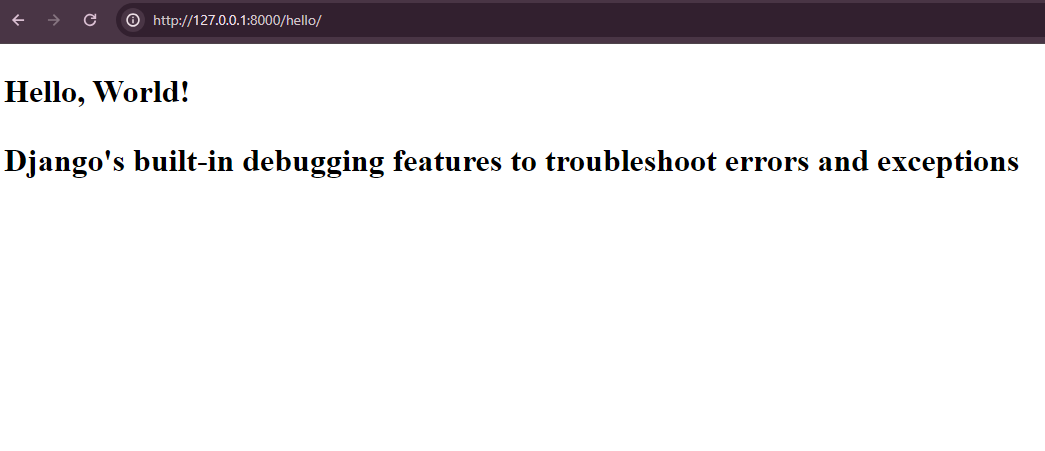
* **Debugging Techniques:**

1. When you access the page, the server will pause at the error line.
2. Use the Django Debug Toolbar (if installed) to inspect queries, templates, and more.
3. Alternatively, add import pdb; pdb.set\_trace() to your view and run the server again. It will drop you into the Python debugger.
4. Explore variables, step through code, and identify issues.

**Remember to remove the error line (my\_variable = 1 / 0) once you’re done debugging**







**WITHOUT EXCEPTION**

**4. A program that creates a web application that implements user authentication and authorization.**

* **Set up Django project and app:**

Follow the steps to create a Django project named myproject and an app named myapp as described in the previous response.

* **Create templates:**

Inside the myapp directory, create a templates directory if it doesn't exist. Then, create login.html and register.html templates for the login and register pages respectively.

**login.html**

<!-- login.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Login</title>

</head>

<body>

    <h2>Login</h2>

    <form method="post">

        {% csrf\_token %}

        <label for="username">Username:</label>

        <input type="text" id="username" name="username">

        <br>

        <label for="password">Password:</label>

        <input type="password" id="password" name="password">

        <br>

        <input type="submit" value="Login">

    </form>

</body>

</html>

**<!-- register.html -->**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Register</title>

</head>

<body>

    <h2>Register</h2>

    <form method="post">

        {% csrf\_token %}

        <label for="username">Username:</label>

        <input type="text" id="username" name="username">

        <br>

        <label for="password1">Password:</label>

        <input type="password" id="password1" name="password1">

        <br>

        <label for="password2">Confirm Password:</label>

        <input type="password" id="password2" name="password2">

        <br>

        <input type="submit" value="Register">

    </form>

</body>

</html>

* **Set up URLs:**

Define URL patterns in **myapp/urls.py** to point to the views for login and register pages:

# myapp/urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('login/', views.login\_view, name='login'),

    path('register/', views.register\_view, name='register'),

]

* **Create views:**

Define views in **myapp/views.py** to render the login and register templates:

# myapp/views.py

from django.shortcuts import render

def login\_view(request):

    return render(request, 'login.html')

def register\_view(request):

    return render(request, 'register.html')

* **Include app URLs:**

Include **myapp** URLs in the **project's** main **urls.py** file:

# myproject/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('myapp.urls')),

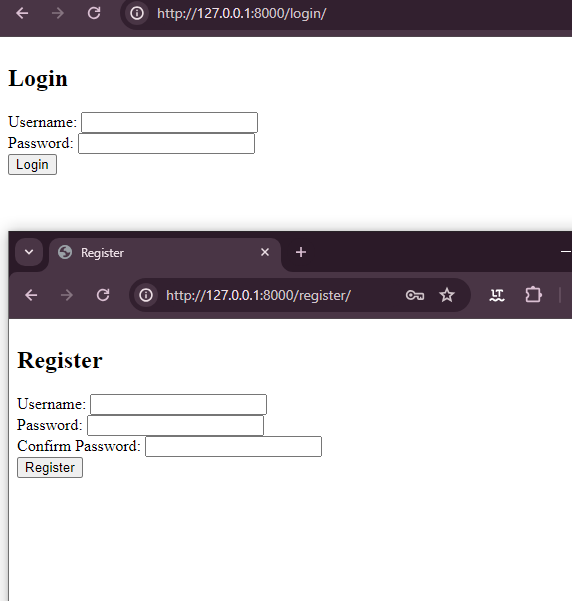
]

* **Run the development server:**

Start the development server:

python manage.py runserver

**OUTPUT:**

****

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

* **Set up Django project and app:**

Start by creating a Django project and app as explained earlier.

* **Install requests library:**

Install the requests library to make HTTP requests to the third-party API:

--------------------------------------🡪 **pip install requests**

* **Integrate with the third-party API:**

Find a suitable third-party API that provides the functionality you need. For example, let's integrate with the OpenWeatherMap API to fetch weather data.

* **Get API key:**

Sign up for an account on the third-party API provider's website and get an API key. For OpenWeatherMap, you can sign up at https://home.openweathermap.org/users/sign\_up and get your API key.

* **Create a view:**

Define a view in views.py to handle the API request and render the fetched data in a template:

# myapp/views.py

from django.shortcuts import render

import requests

def weather\_view(request):

    api\_key = 'YOUR\_API\_KEY'  # Replace 'YOUR\_API\_KEY' with your actual API key

    city = 'New York'  # Example city

    url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}&units=metric'

    response = requests.get(url)

    data = response.json()

    weather\_data = {

        'city': city,

        'temperature': data['main']['temp'],

        'description': data['weather'][0]['description'],

        'icon': data['weather'][0]['icon']

    }

    return render(request, 'weather.html', {'weather\_data': weather\_data})

* **Create a template:**

Create a template weather.html in the templates directory to display the weather data:

<!-- weather.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Weather</title>

</head>

<body>

    <h2>Weather in {{ weather\_data.city }}</h2>

    <p>Temperature: {{ weather\_data.temperature }}°C</p>

    <p>Description: {{ weather\_data.description }}</p>

    <img src="http://openweathermap.org/img/w/{{ weather\_data.icon }}.png" alt="Weather Icon">

</body>

</html>

* **Define URL pattern:**

Define a URL pattern in **urls.py** to map the view:

# myapp/urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('weather/', views.weather\_view, name='weather'),

]

* **Include app URLs:**

Include **myapp** URLs in the project's main **urls.py** file:

# myproject/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('myapp.urls')),

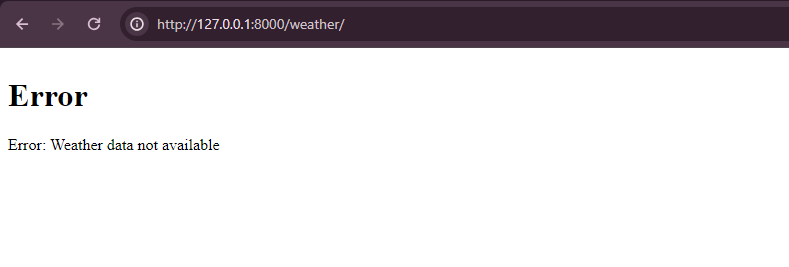
]

* **Run the development server:**

Start the development server:

python manage.py runserver

**OUTPUT:**

****

**SET-5**

**1. A program that creates a simple RESTful API that returns a list of users in JSON format.**

Certainly! Let’s create a simple RESTful API in Python that returns a list of users in JSON format. We’ll use the popular Flask framework to achieve this. Here are the steps:

1. **Setting Up Your Environment**:
   * Make sure you have Python installed on your system.
     + **py –m venv venv**
   * Install Flask using pip install flask.
2. **Create a Python Script**:
   * Create a new Python file (e.g., app.py).
   * Open the file in your favourite code editor.
3. **Write the API Code**:

**# app.py**

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

# Dummy list of users

users = [

    {"id": 1, "name": "John"},

    {"id": 2, "name": "Alice"},

    {"id": 3, "name": "Bob"}

]

# Route to get list of users

@app.route('/users', methods=['GET'])

def get\_users():

    return jsonify(users)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

1. **Explanation**:
   * We’ve defined a simple Flask application with a single route (/users).
   * When a GET request is made to /users, the get users function returns the list of users in JSON format.
2. **Testing the API**:
   * Run your Flask app using python app.py.
   * Open your web browser or use a tool like Postman to send a GET request to http://localhost:5000/users.
   * You should receive a JSON response containing the sample user data.
3. **Customize and Extend**:
   * You can customize the users list with your own data (e.g., from a database).
   * Add more routes and functionality as needed for your specific use case.

**Just click the Link:**

\* Serving Flask app 'app'

\* Debug mode: on

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

\* Running on **http://127.0.0.1:5000**

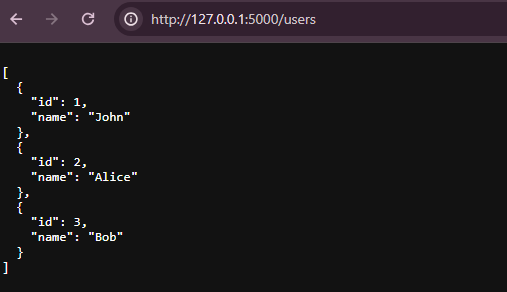
------🡪Press CTRL+C to quit

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 829-761-758

**OUTPUT:**

****

**2. A program that creates a RESTful API that allows users to create, read, update, and delete resources.**

Certainly! Let’s create a simple RESTful API in Python using the Flask framework that returns a list of users in JSON format. Here’s a step-by-step guide:

**1. Setting up Your Environment:**

* Ensure you have Python installed on your system.
* Install Flask using pip install flask.

**2. Create a Python Script:**

* Create a new Python file (e.g., app.py).
* Open the file in your favourite code editor.

**3. Write the API Code**

# app.py

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

# Dummy database to store resources

resources = {

"1": {"id": 1, "Ram": "Resource 1", "Student": "This is my lord Ram.I am your humble servant."},

"2": {"id": 2, "Mannav": "Resource 2", "Student": "This is my classmate."},

"3": {"id": 3, "Jay": "Resource 3", "Student": "This is my best friend."},

"4": {"id": 4, "Saurbh": "Resource 4", "Student": "This is my friend Saurbh."},

"5": {"id": 5, "Mrunalini": "Resource 5", "Student": "This is my friend Mrunalini."},

"6": {"id": 6, "Anmol": "Resource 6", "Student": "This is my friend Anmol."},

"7": {"id": 7, "Shivraj": "Resource 7", "Student": "This is my friend Shivraj."}

}

# Route to create a new resource

@app.route('/resource', methods=['POST'])

def create\_resource():

    data = request.get\_json()

    resource\_id = data.get('id')

    resource\_data = data.get('data')

    resources[resource\_id] = resource\_data

    return jsonify({'message': 'Resource created successfully'}), 201

# Route to read all resources

@app.route('/resources', methods=['GET'])

def get\_all\_resources():

    return jsonify(resources)

# Route to read a specific resource

@app.route('/resource/<int:resource\_id>', methods=['GET'])

def get\_resource(resource\_id):

    if resource\_id in resources:

        return jsonify(resources[resource\_id])

    else:

        return jsonify({'error': 'Resource not found'}), 404

# Route to update a resource

@app.route('/resource/<int:resource\_id>', methods=['PUT'])

def update\_resource(resource\_id):

    if resource\_id in resources:

        data = request.get\_json()

        resources[resource\_id] = data.get('data')

        return jsonify({'message': 'Resource updated successfully'})

    else:

        return jsonify({'error': 'Resource not found'}), 404

# Route to delete a resource

@app.route('/resource/<int:resource\_id>', methods=['DELETE'])

def delete\_resource(resource\_id):

    if resource\_id in resources:

        del resources[resource\_id]

        return jsonify({'message': 'Resource deleted successfully'})

    else:

        return jsonify({'error': 'Resource not found'}), 404

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**4. Explanation:**

* We’ve defined a Flask application with several routes (/resources and /resources/<resource\_id>).
* The routes handle GET, POST, PUT, and DELETE requests for managing resources.
* Replace the resources list with your actual resource data (e.g., from a database).

**5. Testing the API:**

* Run your Flask app using python app.py.
* Use tools like Postman to test the different endpoints (GET, POST, PUT, DELETE).

**6. Customize and Extend:**

* **A**dd authentication, error handling, and other features as needed for your specific use case.

**Just click the Link:**

\* Serving Flask app 'app'

\* Debug mode: on

**WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.**

\* Running on **http://127.0.0.1:5000**

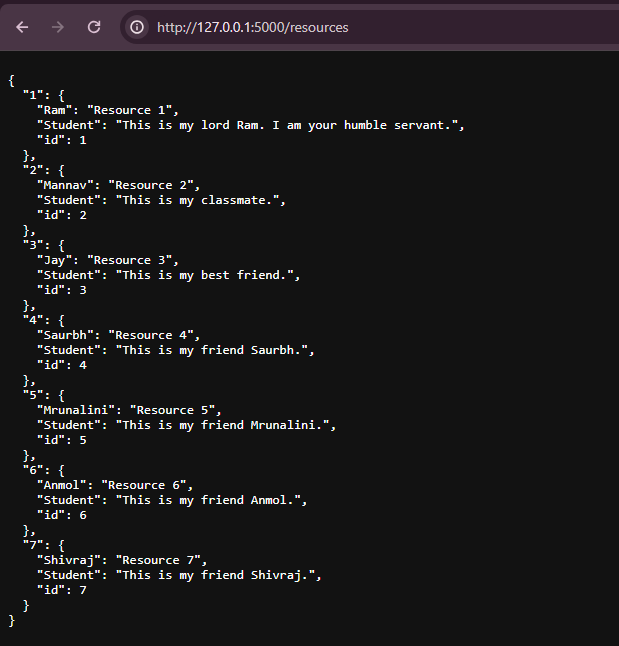
🡪Press CTRL+C to quit

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 829-761-758

**OUTPUT:**

****

**3. A program that creates a RESTful API that authenticates users using a JSON Web Token.**

Creating a RESTful API with JSON Web Token (JWT) authentication in Python is a common task. Let’s break it down into steps:

1. **Setting Up Dependencies:**
   * First, ensure you have Python 3.6 or higher installed.
   * Create a virtual environment (optional but recommended).
   * Install the necessary packages using pip
2. **Program : app.py**

from flask import Flask, render\_template, request, jsonify

import jwt

import datetime

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'your\_secret\_key'

# Mock user data (replace this with your actual user database)

users = {

    'john': 'password1',

    'susan': 'password2'

}

# Function to generate JWT token

def generate\_token(username):

    payload = {

        'username': username,

        'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=30)  # Token expiration time

    }

    token = jwt.encode(payload, app.config['SECRET\_KEY'], algorithm='HS256')

    return token

# Login route

@app.route('/login', methods=['GET', 'POST'])

def login():

    if request.method == 'POST':

        username = request.form['username']

        password = request.form['password']

        if username in users and users[username] == password:

            token = generate\_token(username)

            return jsonify({'token': token.decode('UTF-8')})

        return jsonify({'message': 'Invalid credentials'}), 401

    return render\_template('login.html')

# Protected route

@app.route('/protected', methods=['GET'])

def protected():

    token = request.args.get('token')

    if not token:

        return jsonify({'message': 'Token is missing'}), 401

    try:

        data = jwt.decode(token, app.config['SECRET\_KEY'], algorithms=['HS256'])

        return jsonify({'message': f'Hello, {data["username"]}! You are accessing a protected endpoint.'})

    except jwt.ExpiredSignatureError:

        return jsonify({'message': 'Token has expired'}), 401

    except jwt.InvalidTokenError:

        return jsonify({'message': 'Invalid token'}), 401

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Just click the Link:**

\* Serving Flask app 'app'

\* Debug mode: on

**WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.**

\* Running on **http://127.0.0.1:5000**

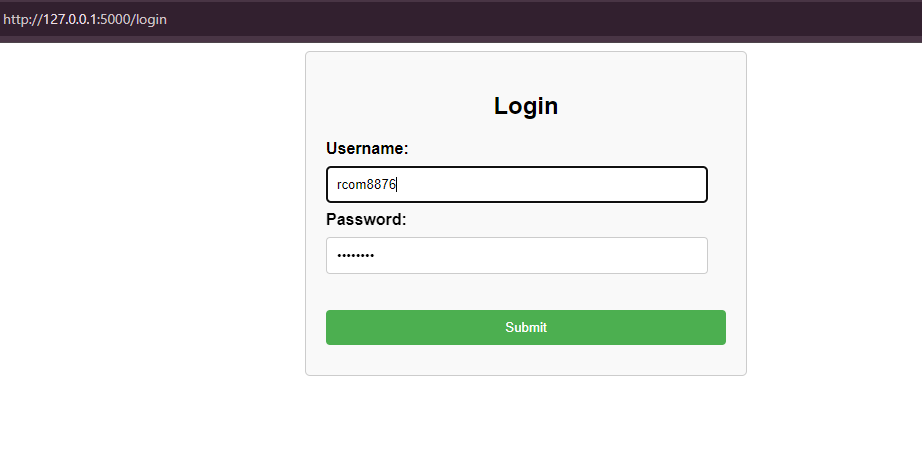
🡪Press CTRL+C to quit

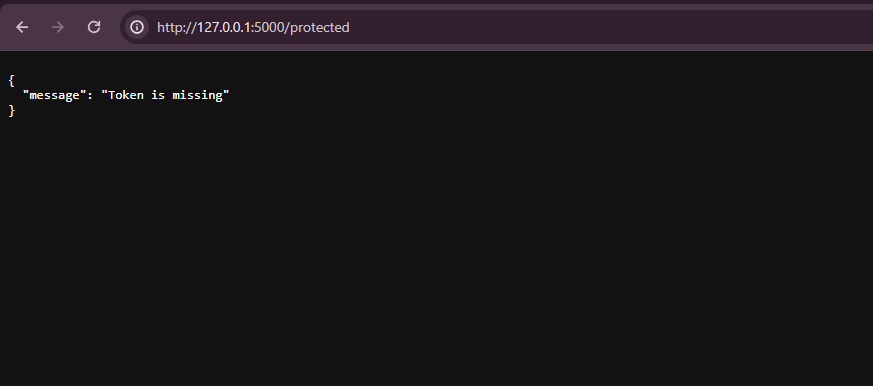
\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 829-761-758

**OUTPUT:**

****

****

**4. A program that creates a RESTful API that paginates the results of a query to improve performance.**

Creating a **RESTful API** that efficiently handles pagination is essential for improving performance and providing a smooth user experience. Let’s break down the steps to achieve this:

**1. Design Your API Endpoints**:

* Define the endpoints for your API. Consider the resources you want to expose (e.g., products, users, posts).
* Plan how you’ll structure your URLs, such as /products, /users, or /posts.

**2. Filtering, Sorting, and Pagination**:

* **Filtering**: Allow clients to narrow down results by specific criteria (e.g., price range, category, date).
* **Sorting**: Enable clients to order results (ascending or descending) based on certain fields (e.g., name, date, and popularity).
* **Pagination**: Divide large result sets into smaller chunks to reduce data transfer and improve performance.

**3. Pagination Techniques**:

* There are several pagination techniques you can choose from:
* **Offset Pagination**: Use offset and limit parameters to specify the starting point and the number of items per page.
* **Cursor-based Pagination**: Use a unique cursor (e.g., an ID or timestamp) to paginate through results.
* **Keyset Pagination**: Rely on the values of specific fields (e.g., timestamps, IDs) to determine the next page.
* **Seek Pagination**: Combine cursor-based and keyset approaches for efficient navigation.

**4. Implementing Offset Pagination (Example)**:

* Let’s create a simple Python program using Flask to demonstrate offset pagination. Install Flask if you haven’t already:

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

# Sample data (replace this with your actual dataset)

data = [

    {'id': 1, 'name': 'Item 1'},

    {'id': 2, 'name': 'Item 2'},

    {'id': 3, 'name': 'Item 3'},

    {'id': 4, 'name': 'Item 4'},

    {'id': 5, 'name': 'Item 5'},

    {'id': 6, 'name': 'Item 6'},

    {'id': 7, 'name': 'Item 7'},

    {'id': 8, 'name': 'Item 8'},

    {'id': 9, 'name': 'Item 9'},

    {'id': 10, 'name': 'Item 10'},

    # Add more data as needed

]

# Paginate the data

def paginate\_data(data, page, per\_page):

    start\_index = (page - 1) \* per\_page

    end\_index = start\_index + per\_page

    paginated\_data = data[start\_index:end\_index]

    return paginated\_data

# Route for fetching paginated data

@app.route('/items', methods=['GET'])

def get\_items():

    page = request.args.get('page', default=1, type=int)

    per\_page = request.args.get('per\_page', default=5, type=int)

    paginated\_data = paginate\_data(data, page, per\_page)

    return jsonify(paginated\_data)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Just click the Link:**

\* Serving Flask app 'app'

\* Debug mode: on

**WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.**

\* Running on **http://127.0.0.1:5000**

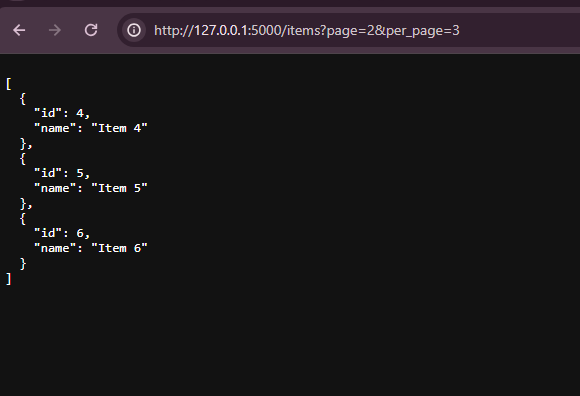
🡪Press CTRL+C to quit

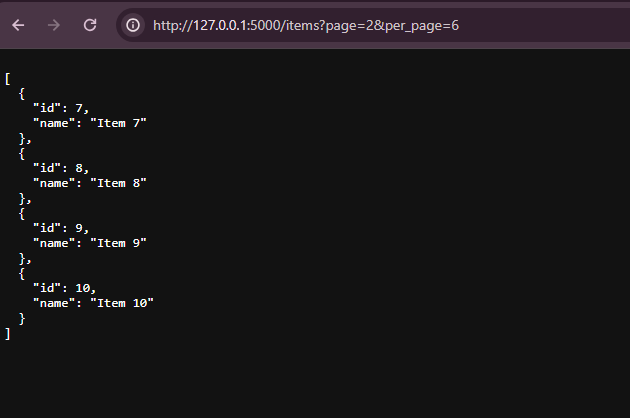
\* Restarting with stat

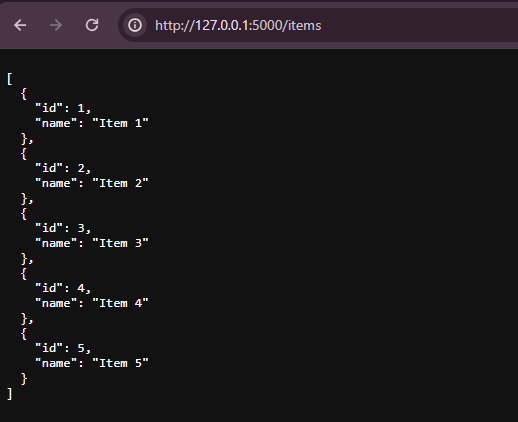
\* Debugger is active!

\* Debugger PIN: 829-761-758

**OUTPUT:**







**5. A program that creates a RESTful API that supports data validation and error handling.**

**RESTful API with Data Validation and Error Handling**

**1. Overview:**

* Python program using Flask.
* Creates a RESTful API supporting CRUD operations (Create, Read, Update, Delete) for user data.

**2. Features:**

* **GET Request to /users:**
  + Retrieves all users from the dataset.
* **GET Request to /users/<user\_id>:**
  + Retrieves a specific user by ID.
  + Example **/user/1**
* **POST Request to /users:**
  + Adds a new user to the dataset.

**3. Data Validation:**

* Validates user input to ensure it contains both name and age fields.
* Checks data types: name must be a string and age must be an integer.

**4. Error Handling:**

* **404 Not Found:**
* Handles requests for non-existent users.
* Returns a JSON response with a 404 error.
* **400 Bad Request:**
* Handles requests with invalid user data.
* Returns a JSON response with a 400 error.

**5. Sample Usage:**

* Test the API using cURL, Postman, or any HTTP client.
* Include JSON data in the request body for adding new users.

**6. Sample Output:**

* JSON responses for successful requests and error cases.
* Responses include user data or appropriate error messages.

**7. Note:**

* Replace sample data with your actual dataset.
* Customize routes, data validation rules, and error handling as needed.

**\*\*\* App.py\*\*\***

from flask import Flask, jsonify, request, abort

app = Flask(\_\_name\_\_)

# Sample data (replace this with your actual dataset)

users = {

    1: {'name': 'Ram', 'age': 21},

    2: {'name': 'Shiv', 'age': 19},

    3: {'name': 'Saurbh', 'age': 20},

    4: {'name': 'Ishita', 'age': 30},

    5: {'name': 'Shivam', 'age': 26},

    6: {'name': 'Pashant', 'age': 27},

}

# Function to validate user input

def validate\_user\_data(data):

    if 'name' not in data or 'age' not in data:

        return False

    if not isinstance(data['name'], str) or not isinstance(data['age'], int):

        return False

    return True

# Route to get all users

@app.route('/users', methods=['GET'])

def get\_users():

    return jsonify(users)

# Route to get a specific user by ID

@app.route('/users/<int:user\_id>', methods=['GET'])

def get\_user(user\_id):

    user = users.get(user\_id)

    if not user:

        abort(404)  # User not found

    return jsonify(user)

# Route to add a new user

@app.route('/users', methods=['POST'])

def add\_user():

    data = request.json

    if not validate\_user\_data(data):

        abort(400)  # Bad request - Invalid user data

    user\_id = max(users.keys()) + 1

    users[user\_id] = {'name': data['name'], 'age': data['age']}

    return jsonify(users[user\_id]), 201  # Created

# Error handler for 404 Not Found

@app.errorhandler(404)

def not\_found(error):

    return jsonify({'error': 'Not Found'}), 404

# Error handler for 400 Bad Request

@app.errorhandler(400)

def bad\_request(error):

    return jsonify({'error': 'Bad Request'}), 400

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Just click the Link:**

\* Serving Flask app 'app'

\* Debug mode: on

**WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.**

\* Running on **http://127.0.0.1:5000**

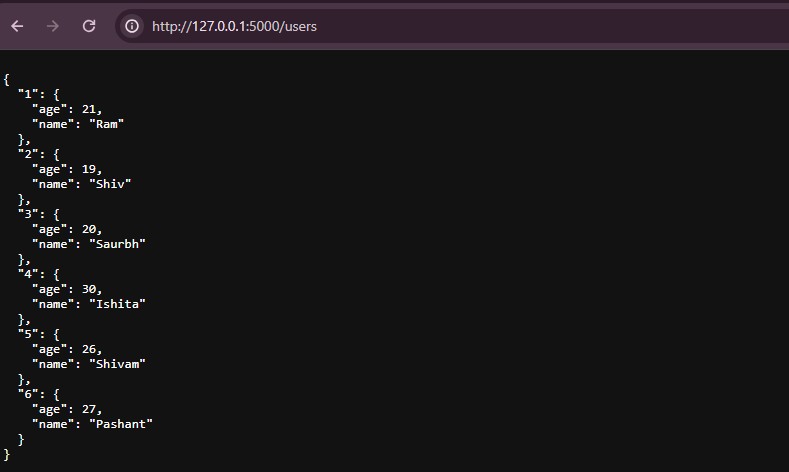
🡪Press CTRL+C to quit

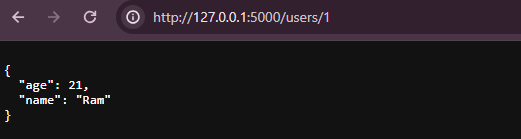
\* Restarting with stat

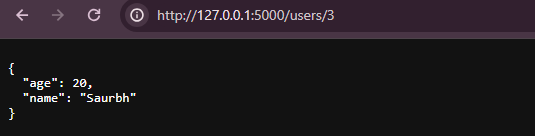
\* Debugger is active!

\* Debugger PIN: 829-761-758

**OUTPUT:**



****

****