Python Flask

Introduction

- Flask is a web framework that provides libraries to build lightweight web applications in python.
- It is developed by **Armin Ronacher** who leads an international group of python enthusiasts (POCCO).
- It is based on WSGI toolkit and jinja2 template engine.
- Flask is considered as a micro framework.

WSGI

- It is an acronym for web server gateway interface which is a standard for python web application development.
- It is considered as the specification for the universal interface between the web server and web application.

Jinja

- Jinja2 is a web template engine which combines a template with a certain data source to render the dynamic web pages.

First Flask application

```
# The file named as Flask.py.
from flask import Flask

app = Flask(__name__)  # creating the Flask class object


@app.route('/')  # decorator drfines the
def home():
    return "Internet of Things";


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

The result in command line .

```
Flask
E:\projectnew11\venv\Scripts\python.exe E:/projectnew11/Flask.py
 * Serving Flask app 'Flask' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 120-807-968
127.0.0.1 - - [31/Oct/2023 05:16:51] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Oct/2023 05:16:55] "GET /favicon.ico HTTP/1.1" 404 -
```

A web application, run on the browser at http://localhost:5000.

- To build the python web application, we need to import the Flask module.
- An object of the Flask class is considered as the WSGI application.
- We need to pass the name of the current module, i.e. __name__ as the argument into the Flask constructor.
- The route() function of the Flask class defines the URL mapping of the associated function. The syntax is given below.

app.route(rule, options)

It accepts the following parameters.

- **rule:** It represents the URL binding with the function.
- **options:** It represents the list of parameters to be associated with the rule object

- As we can see here, the / URL is bound to the main function which is responsible for returning the server response. It can return a string to be printed on the browser's window or we can use the HTML template to return the HTML file as a response from the server.
- Finally, the run method of the Flask class is used to run the flask application on the local development server.

- Finally, the run method of the Flask class is used to run the flask application on the local development server.

The syntax is given below.

app.run(host, port, debug, options)

SN	Option	Description
1	host	The default hostname is 127.0.0.1, i.e. localhost.
2	port	The port number to which the server is listening to. The default port number is 5000.
3	debug	The default is false. It provides debug information if it is set to true.
4	options	It contains the information to be forwarded to the server.

Flask App routing

- App routing is used to map the specific URL with the associated function that is intended to perform some task. It is used to access some particular page in the web application.

- In our first application, the URL ('/') is associated with the home function that returns a particular string displayed on the web page.

- In other words, we can say that if we visit the particular URL mapped to some particular function, the output of that function is rendered on the browser's screen.

- In flask, the URL (“/”) is associated with the root URL. So if our site’s domain was www.example.org and we want to add routing to “www.example.org/hello”, we would use “/hello”.

- To bind a function to an URL path we use the app.route decorator.

- In the below example, we have implemented the above routing in the flask.

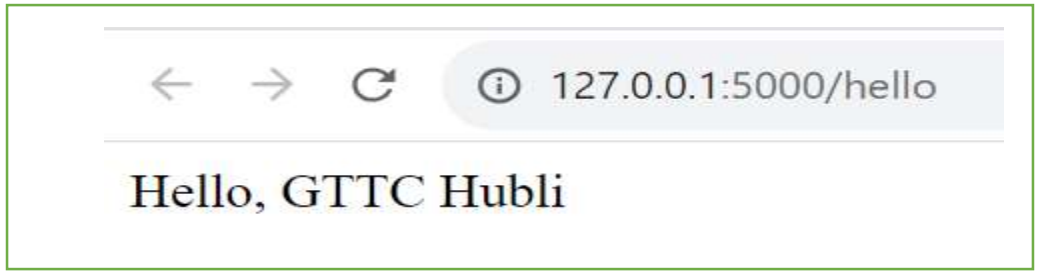
main.py

```
from flask import Flask

app = Flask(__name__)

# Pass the required route to the decorator.
@app.route("/hello")
def hello():
    return "Hello, GTTC Hubli"

@app.route("/")
def index():
    return "Homepage of GTTC"

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

RESULT

- The hello function is now mapped with the "/hello" path and we get the output of the function rendered on the browser.

- Open the browser and visit *127.0.0.1:5000/hello*, you will see the following output.



- Flask facilitates us to add the variable part to the URL by using the section. We can reuse the variable by adding that as a parameter into the view function.

- The converter can also be used in the URL to map the specified variable to the particular data type. For example, we can provide the integers or float like age or salary respectively.

- The following converters are used to convert the default string type to the associated data type.
  1. **string**: default
  2. **int:** used to convert the string to the integer
  3. **float:** used to convert the string to the float.
  4. **path:** It can accept the slashes given in the URL.
  5. **uuid:** It accepts UUID strings.

Example 1:

```
from flask import Flask
app = Flask(__name__)

@app.route('/course/<name>')
def course(name):
    # Course Name
    return ‘COURSE ,’+ name

@app.route("/hello")
def hello():
    return "Hello, GTTC Hubli"

@app.route("/")
def index():
    return "Homepage of GTTC"

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

OUTPUT:

Example 2

```
from flask import Flask
app = Flask(__name__)

@app.route('/ID/<int:std_id>')
def id(std_id):
    return "Student ID is = %d"%std_id;

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

OUTPUT:

The add_url_rule() function

- There is one more approach to perform routing for the flask web application that can be done by using the add_url() function of the Flask class.

- The syntax to use this function is

  add_url_rule(<url rule>, <endpoint>, <view function>)

- This function is mainly used in the case if the view function is not given and we need to connect a view function to an endpoint externally by using this function.

Example:

```
from flask import Flask

app = Flask(__name__)

def about():
    return "Multi Skill Development Center"

app.add_url_rule("/about", "about", about)

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

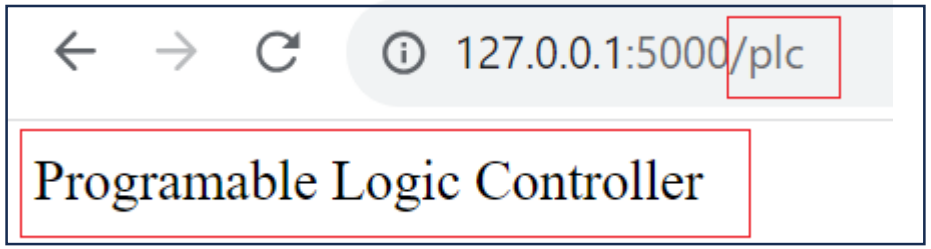
OUTPUT:

Flask URL Building

- The url_for() function is used to build a URL to the specific function dynamically. The first argument is the name of the specified function, and then we can pass any number of keyword argument corresponding to the variable part of the URL.

- This function is useful in the sense that we can avoid hard-coding the URLs into the templates by dynamically building them using this function.

```python
from flask import *
app = Flask(__name__)

@app.route('/')
def dept():
    return 'MSDC'

@app.route('/iot')
def iot():
    return 'Internet of Things'

@app.route('/plc')
def plc():
    return 'Programable Logic Controller'

@app.route('/cad')
def cad():
    return 'Computer Adied Design'

@app.route('/user/<name>')
def user(name):
    if name == 'iot':
        return redirect(url_for('iot'))
    if name == 'plc':
        return redirect(url_for('plc'))
    if name == 'cad':
        return redirect(url_for('cad'))

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



Accura Tequipment

Flask HTTP methods

- HTTP is the hypertext transfer protocol which is considered as the foundation of the data transfer in the world wide web. All web frameworks including flask need to provide several HTTP methods for data communication.

- We can specify which HTTP method to be used to handle the requests in the route() function of the Flask class. By default, the requests are handled by the GET() method.

The methods are given in the following table

SN	Method	Description
1	GET	It is the most common method which can be used to send data in the unencrypted form to the server.
2	HEAD	It is similar to the GET but used without the response body.
3	POST	It is used to send the form data to the server. The server does not cache the data transmitted using the post method.
4	PUT	It is used to replace all the current representation of the target resource with the uploaded content.
5	DELETE	It is used to delete all the current representation of the target resource specified in the URL.

Flask HTTP GET Method

```
from flask import Flask, request, render_template
app = Flask(__name__)
# Initialize an empty list to store user data
users = []
@app.route('/')
def data():
    return render_template('GetForm.html')

@app.route('/register', methods=['GET'])
def signin():
    # Access form data using request.form instead of request.args for a POST request
    fname = request.args.get('fname')
    dob = request.args.get('dob')
    age = request.args.get('age’)
```

Continued…….

```
 user = []
    user.append(fname)
    user.append(dob)
    user.append(age)
   # Store user data in the users list
    users.append(user)
    # Print user data (for testing purposes)
    print(fname, dob, age)
    # Pass the users list to the template
    return render_template('GetForm.html', users=users)

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

Example: GetForm.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Using GET Method</title>
   <style>
      table, td, th{border: 1px solid black;}
   </style>
</head>
<body>
<h1>GET Method form</h1>
<form action="/register" method="GET">
   <input type="text" id="fname" name="fname" placeholder="First Name"><br><br>
    <input type="date" id="dob" name="dob" placeholder="Date of Birth"><br><br>
    <input type="number" id="age" name="age" placeholder="Age" min="18" max="35"><br><br>
   <input type="submit" value="Submit"><br><br>
</form>
```

Continued……

Continued……

```
<table style="width:70%">
   <tr>
      <th>Name</th>
      <th>date of Birth</th>
      <th>Age</th>
   </tr>
   {% for row in users  %}
   <tr>
    <td>{{row[0]}}</td>
      <td>{{row[1]}}</td>
      <td>{{row[2]}}</td>
   </tr>
   {% endfor %}
</table>
</body>
</html>
```

Output:

GET Method form

First Name

dd-----yyyy

Age

Submit

Name	date of Birth	Age
Pushpa L M	1988-01-05	35
Kiran K	2000-09-08	23

Flask HTTP POST Method

```
from flask import Flask, request, render_template
app = Flask(__name__)
# Initialize an empty list to store user data
users = []
@app.route('/')
def data():
    return render_template('PostForm.html')

@app.route('/register', methods=['POST'])
def signin():
    # Access form data using request.form instead of request.args for a POST request
    if request.method=="POST":
        fname = request.form['fname']
        dob = request.form['dob']
        age = request.form['age']
        gender=request.form['gender’]
```

Continued…….

Continued……

```
user = [ ]
    user.append(fname)
    user.append(dob)
    user.append(age)
    user.append(gender)

# Store user data in the users list
    users.append(user)

# Print user data (for testing purposes)
    print(fname, dob, age, gender)

    # Pass the users list to the template
    return render_template('PostForm.html', users=users)

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

Example: **PostForm.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Using GET Method</title>
    <style>
        table, td, th{border: 1px solid black;}
    </style>
</head>
<body>
<h1>POST Method form</h1>
<form action="/register" method="POST">
    <input type="text" id="fname" name="fname" placeholder="First Name"><br><br>
     <input type="date" id="dob" name="dob" placeholder="Date of Birth"><br><br>
     <input type="number" id="age" name="age" placeholder="Age" min="18" max="35"><br><br>
    <input type="radio" name="gender" value="Male">Male
        <input type="radio" name="gender" value="Female">FeMale<br><br>
    <input type="submit" value="Submit"><br><br>


</form>
```

Continued……..

Continued……..

```
<table style="width:100%">
    <tr>
        <th>Name</th>
        <th>date of Birth</th>
        <th>Age</th>
        <th>Gender</th>
    </tr>
    {% for row in users  %}
    <tr>
     <td>{{row[0]}}</td>
         <td>{{row[1]}}</td>
         <td>{{row[2]}}</td>
        <td>{{row[3]}}</td>
    </tr>
    {% endfor %}
</table>

</body>
</html>
```

Output:

Accura Tequipment

POST Method form

First Name

dd - - - - - yyyy

Age

Male FeMale

Submit

Name	date of Birth	Age	Gender
Akash V	1990-02-16	33	Male
Deepa Patil	2001-01-05	22	Female
Kavita Desai	1999-02-28	24	Female

Flask Templates

- Flask facilitates us to return the response in the form of HTML templates. Rendering external HTML files
- Flask facilitates us to render the external HTML file instead of hardcoding the HTML in the view function. Here, we can take advantage of the jinja2 template engine on which the flask is based.
- Flask provides us the render_template() function which can be used to render the external HTML file to be returned as the response from the view function.

message.html

#To render an HTML file from the view function, let's first create an HTML file named as message.html.

```
<html>
<head>
<title>Message</title>
</head>
<body>
<h1>hi, welcome to the GTTC   Hubbli </h1>
</body>
</html>
```

script.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def message():
    return render_template('message.html’)

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

- Run the flask script **script.py** and visit *https://localhost:5000* on the browser the result is.

Delimiters

- Jinga 2 template engine provides some delimiters which can be used in the HTML to make it capable of dynamic data representation. The template system provides some HTML syntax which are placeholders for variables and expressions that are replaced by their actual values when the template is rendered.

- The jinga2 template engine provides the following delimiters to escape from the HTML.
  - {% ... %} for statements
  - {{ ... }} for expressions to print to the template output
  - {# ... #} for the comments that are not included in the template output
  - # ... ## for line statements

Flask SQLite

- Flask can make use of the SQLite3 module of the python to create the database web applications.

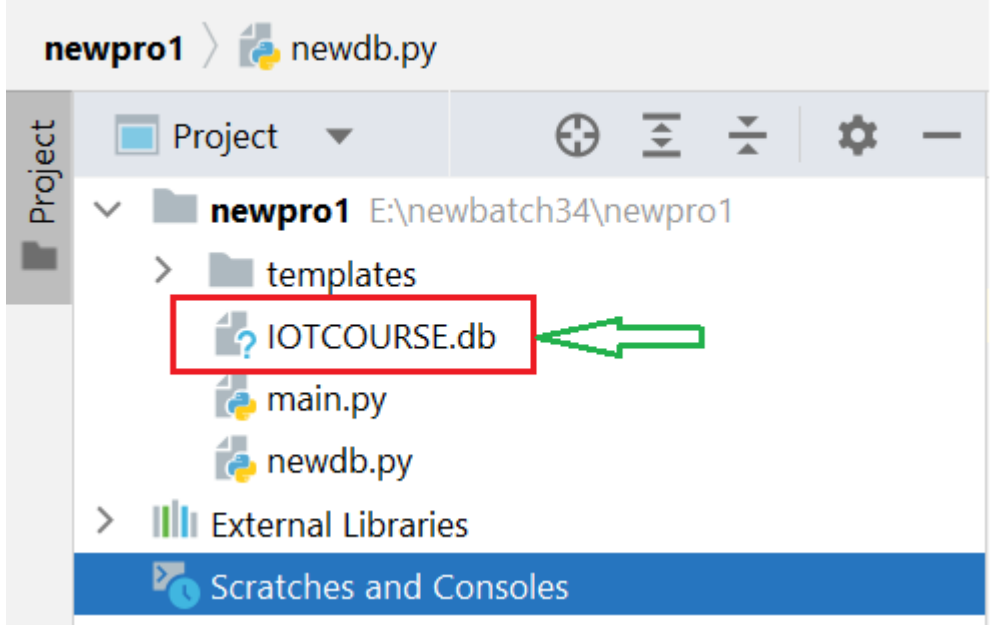
- In Flask using SQLite we can create a CRUD (create - read - update - delete) application.

CRUD Application in flask

- For this purpose, the database should be created **IOTCOURSE.db**

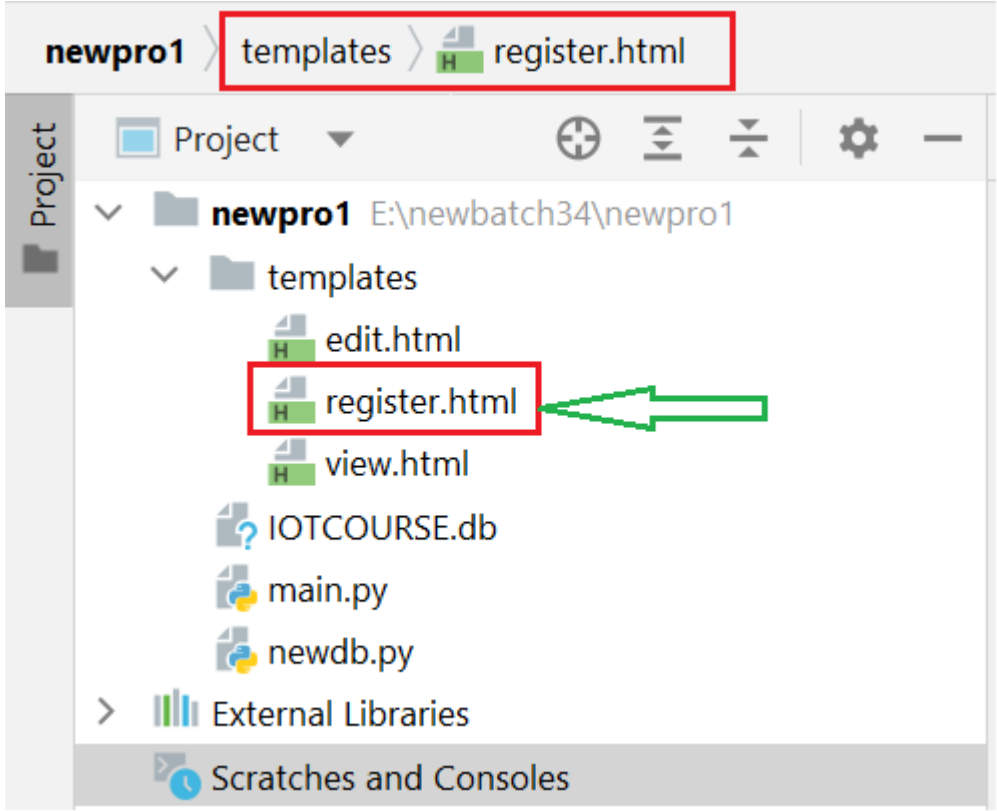
newdb.py

- **Create a data base using python file**




```
import sqlite3
conn = sqlite3.connect("IOTCOURSE.db")
conn.execute("CREATE TABLE Students(id INTEGER PRIMARY KEY AUTOINCREMENT, fname TEXT, mob TEXT, dob TEXT ) ")
```

- Next create the view function: first() which is associated with the URL (/). It renders a template register.html.




```
@app.route('/')
def first():
    return render_template("register.html")
```

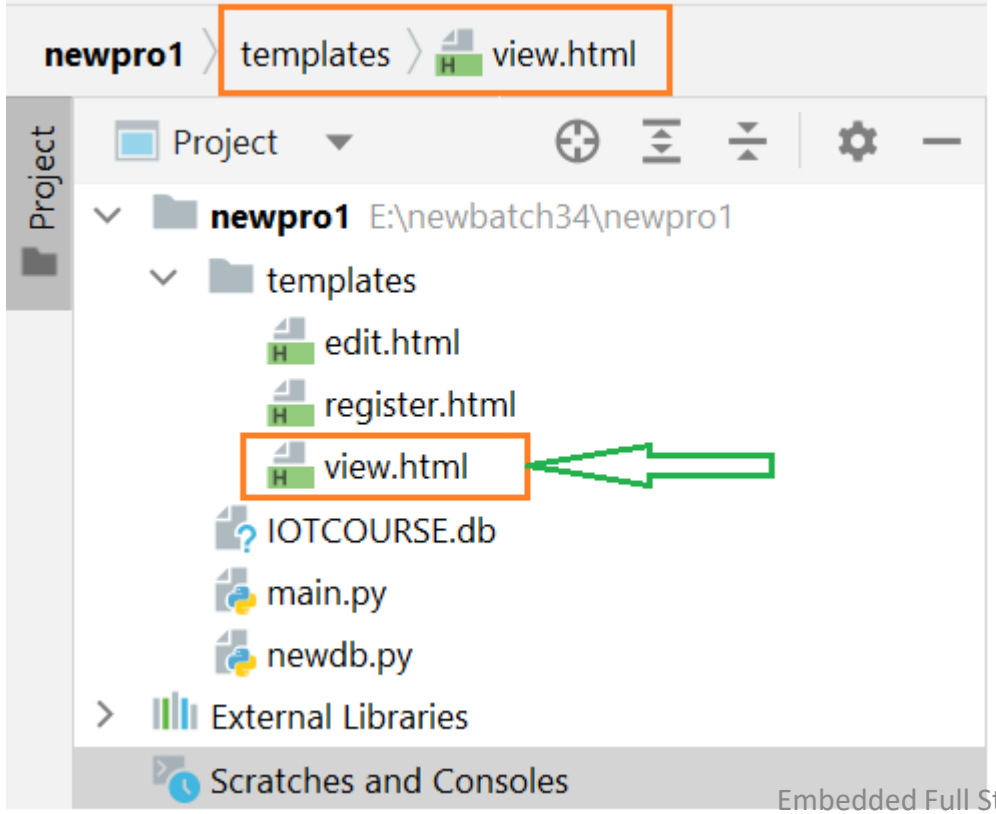
register.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Register Form</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
   <div style="text-align:center">
      <h1 >Form Design</h1>
      <h4 ><a href="/view" >VIEW</a> </h4>
      <h4 >{{msg}}</h4>
      <form action="/register" method="POST">
            First Name: <input type="text" id="fname" name="fname" required><br><br>
            Mobile No : <input type="text" id="mob" name="mob"><br><br>
            Date of Birth: <input type="date" id="dob" name="dob"><br><br>
            <input type="submit" value="Register">
      </form>
   </div>
</body>
</html>
```

- All the details entered by the students is posted to the URL **/register** which is associated with the function **register().** The function **register()** is given below which contains the code for extracting the data entered by the students and save that data into the table Students.

```
@app.route('/register' , methods=["POST"])
def register():
    if request.method == "POST":
        fname = request.form['fname']
        mob = request.form['mob']
        dob = request.form['dob']
        print(fname, mob, dob)
        conn = sqlite3.connect("IOTCOURSE.db")
        conn.execute("INSERT INTO students(fname, mob, dob)VALUES(?,?,?)", (fname, mob, dob ))
        conn.commit()
        msg = "Registered successfully!"
        return render_template("register.html", msg=msg)
```

- The entered details can be stored in database IOTCOURSE.db . The details can be viewed in renders a template **view.html** with the function **view()**




```
@app.route('/view')
def view():
    conn = sqlite3.connect("IOTCOURSE.db")
    cur = conn.execute("SELECT * FROM students")
    rows = cur.fetchall()
    return render_template("view.html", rows=rows)
```

view.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>View Details</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <style>
      table, td, th{border:1px solid black;}
   </style>
</head>
<body>
<div style="text-align:center">
   <h1>View Details</h1>
   <h4 ><a href="/" >Home</a>     <a href="/view" >View
Details</a> </h4>
<table style="width:100%">
   <thead>
   <tr>
      <th>Id</th>
      <th>Name</th>
      <th>Mobile</th>
      <th>DOB</th>
       <th>Action</th>
   </tr>
   </thead>
```

continued……..

continued……..

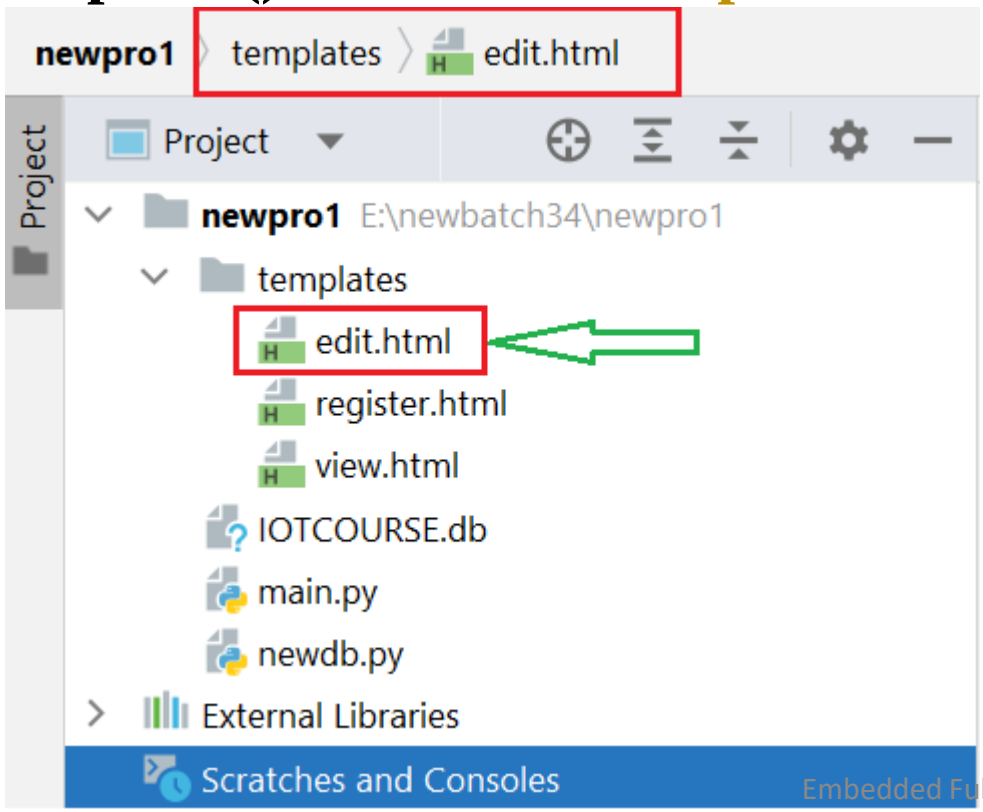
```
<tbody>
    {% for row in rows %}
   <tr>
      <td>{{row[0]}}</td>
      <td>{{row[1]}}</td>
      <td>{{row[2]}}</td>
      <td>{{row[3]}}</td>
      <td><a href="/del/{{row[0]}}">Delete</a>, <a
href="/edit/{{row[0]}}">Edit</a> </td>
   </tr>
   {% endfor %}
   </tbody>
</table>
</div>
</body>
</html>
```

- The ID entered by the student is posted to the URL **/del** which contains the python code to establish the connection to the database and then delete all the records for the specified ID. The URL **/del** is associated with the function **delete()** which is given below.

```
@app.route('/del/<a>')
def delete(a):
    con=sqlite3.connect("IOTCOURSE.db")
    con.execute("DELETE FROM students where id=?",(a,))
    con.commit()
    return redirect(url_for('view’))
```

- After delete() function It returns a response object and redirects the URL /view

- The entered details can be stored in database IOTCOURSE.db . The details can be edited with the URL **/edit** which is associated with the the function **edit() .**
- The details can be updated in renders a template **edit.html** with the function **update()** with the URL **/update**




```
@app.route('/edit/<a>')
def edit(a):
    con=sqlite3.connect("IOTCOURSE.db")
    cur=con.execute("SELECT *FROM students where id=?",(a,))
    row=cur.fetchone()
    return render_template("edit.html", row=row)
```

edit.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>About</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
<div style="text-align:center">
   <h1>Edit Details</h1>
   <h4 ><a href="/view" >View Details</a> </h4>
   <h4 >{{msg}}</h4>
    <form action="/update/{{row[0]}}" method="POST">
         First Name: <input type="text" id="fname" name="fname" value="{{row[1]}}" required><br><br>
         Mobile No : <input type="text" id="mob" name="mob" value="{{row[2]}}"><br><br>
         Date of Birth: <input type="date" id="dob" name="dob" value="{{row[3]}}"><br><br>
         <input type="submit" value="update">
    </form>
</div>
</body>
</html>
```

- The ID entered by the student is posted to the URL **/edit** which contains the python code to establish the connection to the database and then edit all the records for the specified ID. The URL **/update** is associated with the function **update()** which is given below.

```
@app.route('/update/<a>',methods=['POST'])
def update(a):
    if request.method=='POST':
        fname=request.form['fname']
        mob=request.form['mob']
        dob=request.form['dob']
        con=sqlite3.connect('IOTCOURSE.db')
        con.execute('UPDATE students SET fname=?, mob=?,dob=? WHERE id=?',(fname,mob,dob,a))
        con.commit()
        return redirect(url_for('view’))
```

- After update() function It returns a response object and redirects the URL /view

main.py

main.py | view.html | edit.html | register.html | newdb.py

```python
from flask import*
import sqlite3
app=Flask(__name__)

@app.route('/')
def first():
    return render_template("register.html")

@app.route('/register' , methods=["POST"])
def register():
    if request.method == "POST":
        fname = request.form['fname']
        mob = request.form['mob']
        dob = request.form['dob']
        print(fname, mob, dob)
        conn = sqlite3.connect("IOTCOURSE.db")
        conn.execute("INSERT INTO students(fname,mob,dob)VALUES(?,?,?)", (fname, mob, dob ))
        conn.commit()
        msg = "Registered successfully!"
        return render_template("register.html", msg=msg)
```

cont.

```
22  @app.route('/view')
23  def view():
24      conn = sqlite3.connect("IOTCOURSE.db")
25      cur = conn.execute("SELECT * FROM students")
26      rows = cur.fetchall()
27      return render_template("view.html", rows=rows)
28
29  @app.route('/del/<a>')
30  def delete(a):
31      con=sqlite3.connect("IOTCOURSE.db")
32      con.execute("DELETE FROM students where id=?",(a,))
33      con.commit()
34      return redirect(url_for('view'))
35
36  @app.route('/edit/<a>')
37  def edit(a):
38      con=sqlite3.connect("IOTCOURSE.db")
39      cur=con.execute("SELECT *FROM students where id=?",(a,))
40      row=cur.fetchone()
41      return render_template("edit.html", row=row)
```

cont.

```
43
44  @app.route('/update/<a>',methods=['POST'])
45  def update(a):
46      if request.method=='POST':
47          fname=request.form['fname']
48          mob=request.form['mob']
49          dob=request.form['dob']
50          con=sqlite3.connect('IOTCOURSE.db')
51          con.execute('UPDATE students SET fname=?, mob=?,dob=? WHERE id=?',(fname,mob,dob,a))
52          con.commit()
53          return redirect(url_for('view'))
54
55
56  if __name__ == "__main__":
57      app.run()
58
```

main.py

- Run the flask script main.py and visit *https://localhost:5000* on the browser.

```
Run:  main
D:\bordersecurity\newpro1\Scripts\python.exe E:/newbatch34/newpro1/main.py
 * Serving Flask app 'main'
 * Debug mode: off
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
```

- To visit http://127.0.0.1:5000 on browser

Form Design

VIEW

First Name:

Mobile No :

Date of Birth: dd-mm-yyyy

Register

- To visit view on home page

View Details

Home View Details

Id	Name	Mobile	DOB	Action
10	GTTC Hubli	1234554321	2023-11-06	Delete,Edit
11	Asha K	9876556789	2023-11-05	Delete,Edit

- To visit edit in view page

Edit Details

View Details

First Name: MSDC

Mobile No : 1234554321

Date of Birth: 06-11-2023

update

- After row update the result is

View Details

Home View Details

Id	Name	Mobile	DOB	Action
10	MSDC	1234554321	2023-11-06	Delete,Edit
11	Asha K	9876556789	2023-11-05	Delete,Edit

Reference

- https://flask.palletsprojects.com/en/3.0.x/
- https://www.geeksforgeeks.org/flask-tutorial/
- https://pythonbasics.org/what-is-flask-python/
- https://www.javatpoint.com/flask-tutorial