UNIT 4 : FLASK FRAMEWORK

Flask HTTP methods

1. HTTP is the hypertext transfer protocol which is considered as the foundation of the data transfer in the world wide web.
2. All web frameworks including flask need to provide several HTTP methods for data communication.
3. We can specify which HTTP method to be used to handle the requests in the route() function of the Flask class.
4. By default all the requests are handled by method GET().

Following are some methods that are used along with flask framework

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

POST Method

- To handle the POST requests at the server end we first create a form to get some data at the client side from the end user, and then we will try to access that data on the server side by using the POST request.

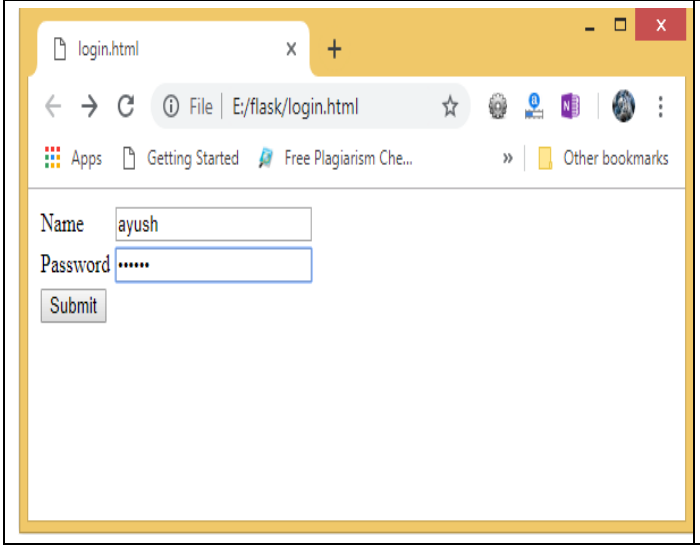
login.html

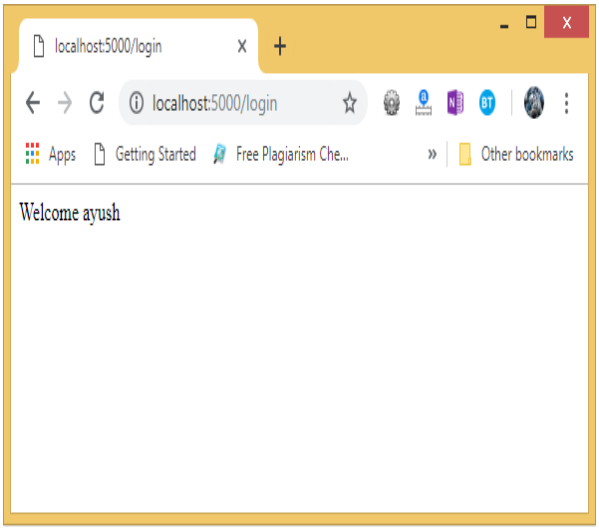
```
<html>
  <body>
      <form action = "http://localhost:5000 /login" method = "post">
      <table>
      <tr><td>Name</td>
      <td><input type ="text" name = "uname"></td></tr>
      <tr><td>Password</td>
      <td><input type ="password" name = "pass"></td></tr>
      <tr><td><input type = "submit"></td></tr>
  </table>
```

post_example.py

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

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

def login():
    uname=request.form['uname']
    passwrd=request.form['pass']
    if uname=="ayush" and passwrd=="google":
        return "Welcome %s" %uname

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

```
    </form>
  </body>
</html>
```





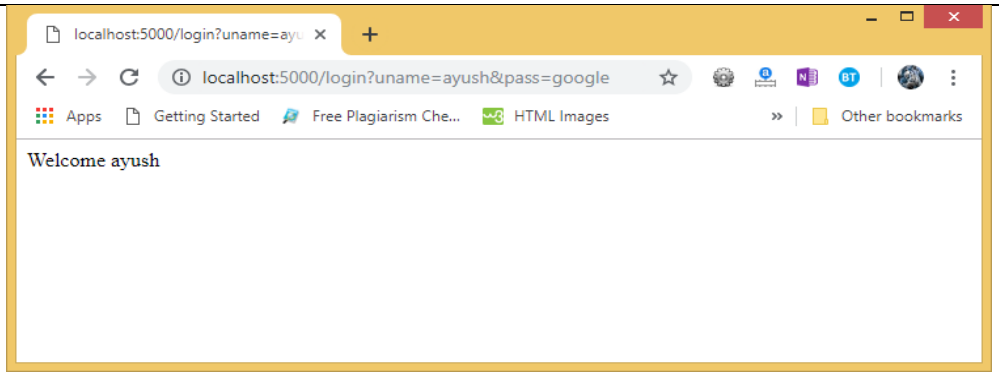
- Enter the code into the script named post_example.py.
- Then we will start the development server by running the script using python post_exmple.py and open login.html on the web browser
- Give the required input and click Submit, we will get the following result.
- Hence, the form data is sent to the development server by using the post method.

GET Method

- With the same example we will understand the GET method but there are some changes in the data retrieval syntax on the server side.
- Login.html will be created as same we created above

get_example.py

```
from flask import *
app = Flask(__name__)
    @app.route('/login',methods = ['GET'])
def login():
    uname=request.args.get('uname')   #CHANGES IN CODE FOR GET
    passwrd=request.args.get('pass')   #CHANGES IN CODE FOR GET
    if uname=="ayush" and passwrd=="google":
        return "Welcome %s" %uname
  if __name__ == '__main__':
  app.run(debug = True)
```



- The data sent using the get() method is retrieved on the development server.
- Check the URL

- uname = request.args.get('uname') is the code by which data is obtained
- The args is a dictionary object which contains the list of pairs of form parameter and its corresponding value.
- the data sent to the server is not shown in the URL on the browser in the POST requests.

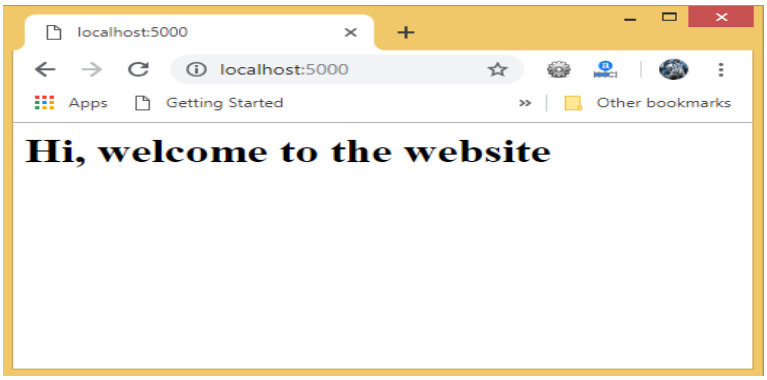
Flask Templates

flask facilitates us to return the response in the form of HTML templates.

- **Instead of returning a simple plain string as a message, it returns a message with <h1> tag attached to it using HTML**

script.py

```
from flask import *
app = Flask(__name__)
@app.route('/')
def message():
    return "<html><body><h1>Hi, welcome to the website</h1></body></html>"
if __name__ == '__main__':
  app.run(debug = True)
```



Flask facilitates us to render the external HTML file instead of hardcoding the HTML in the view function.

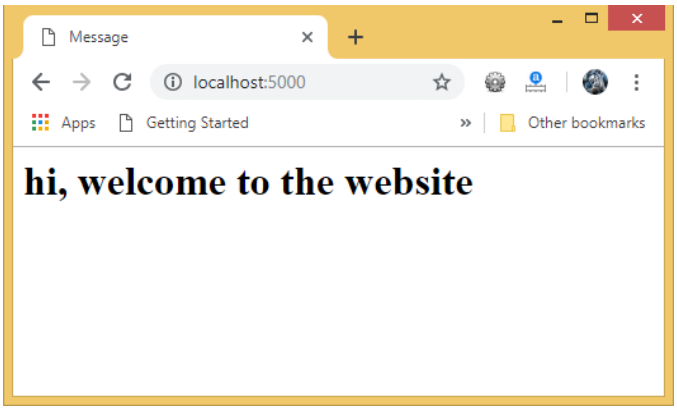
1. We need to use the render_template() function to render external HTML files.
2. we must create the folder **templates** inside the application directory and save the HTML templates referenced in the flask script in that directory.

message.html

```
1. <html>
2. <head>
3. <title>Message</title>
4. </head>
5. <body>
6. <h1>hi, welcome to the website </h1>
7. </body>
8. </html>
```

script.py

```
1. from flask import *
2. app = Flask(__name__)
3.
4. @app.route('/')
5. def message():
6.     return render_template('message.html')
7. if __name__ == '__main__':
8.   app.run(debug = True)
```



Embedding Python statements in HTML

When we may need to execute the statements for the general-purpose computations. Flask facilitates us the delimiter {%...%} which can be used to embed the simple python statements into the HTML.

DESIGN A CODE TO PRINT THE TABLE OF 10

script.py

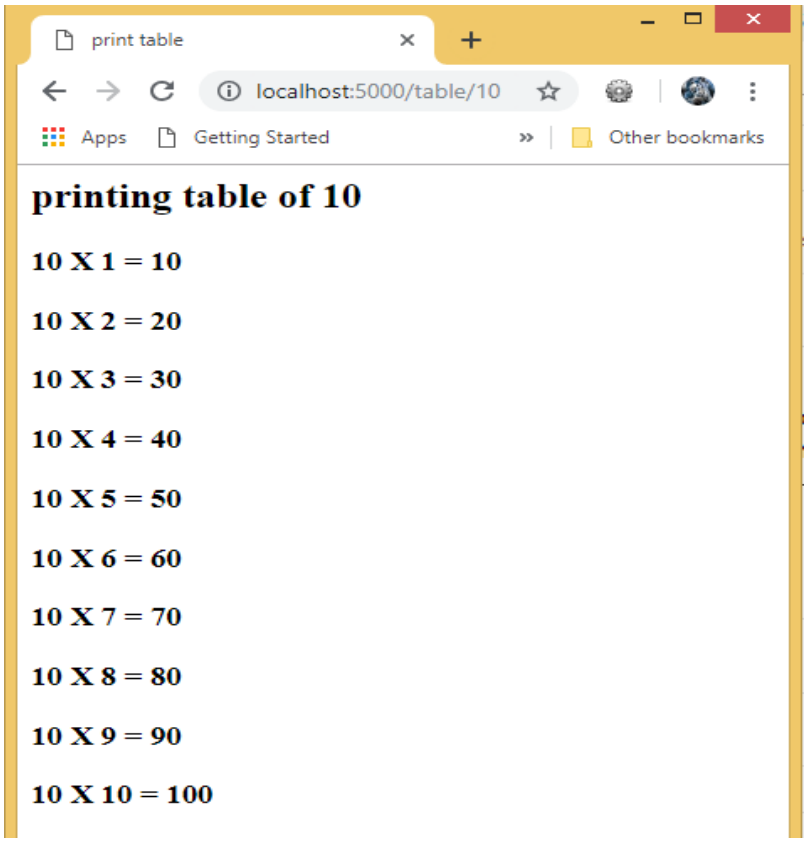
```
1. from flask import *
2. app = Flask(__name__)
3.
4. @app.route('/table/<int:num>')
5. def table(num):
```

print-table.py

```
1. <html>
2. <head>
3. <title>print table</title>
4. </head>
5. <body>
6. <h2> printing table of {{n}}</h2>
```

```
6.      return render_template('print-table.html',n=num)
7. if __name__ == '__main__':
8.     app.run(debug = True)
```

```
7. {% for i  in range(1,11): %}
8.     <h3>{{n}} X {{i}} = {{n * i}} </h3>

9. {% endfor %}
10. </body>
11. </html>
```



A CODE WHICH RETURNS HTML FILE WHICH IS STYLED USING CASCADING STYLE SHEET (CSS)

script.py

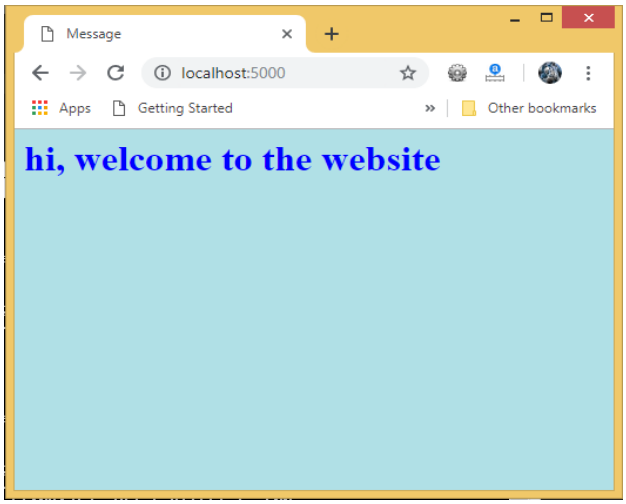
```
1. from flask import *
2. app = Flask(__name__)
3.
4. @app.route('/')
5. def message():
6.      return render_template('message.html')
7. if __name__ == '__main__':
8.     app.run(debug = True)
```

message.html

```
<html>
<head>
    <title>Message</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>

<body>
    <h1>hi, welcome to the website</h1>
</body>
</html>
```

style.css

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```



CREATION OF APPLICATION FORM WHICH ACCEPTS THE USER DETAILS AND CONFIRMS IT AFTER SUBMISSION

- For this is to be done we need three files, i.e., script.py, customer.html, and result_data.html.
- Here we create the print_data() function which collects all the data from the request object and renders the result_data.html file which shows all the data on the web page for confirmation.

script.py

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

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

```

```
@app.route('/success',methods = ['POST', 'GET'])
def print_data():
  if request.method == 'POST':
    result = request.form
    return render_template("result_data.html",result = result)

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

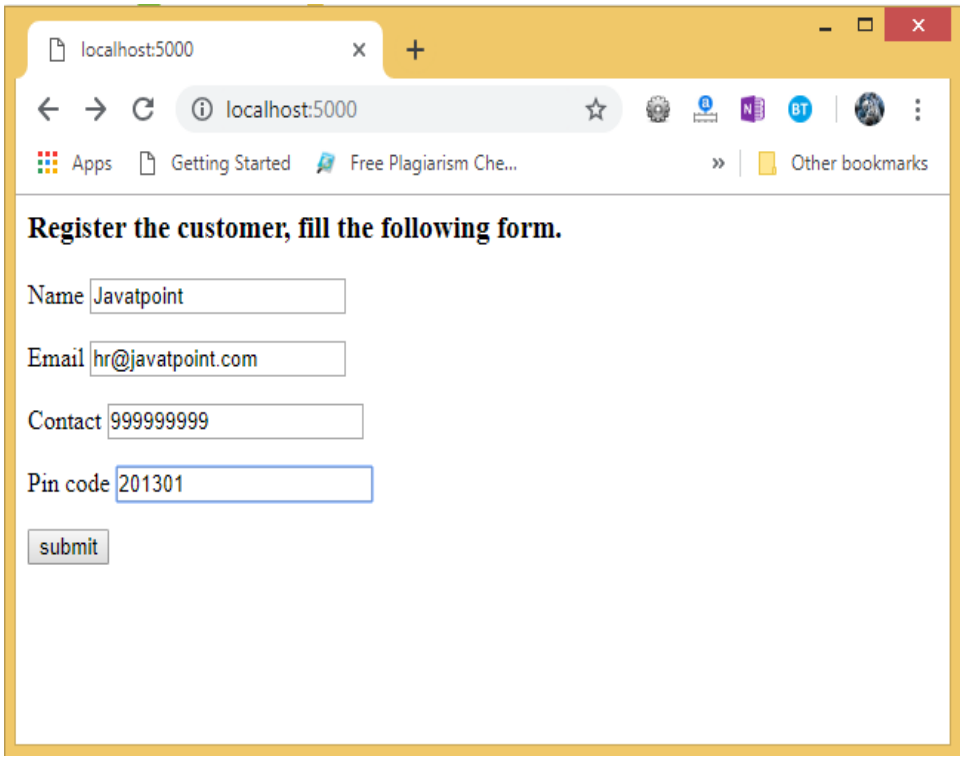
customer.html

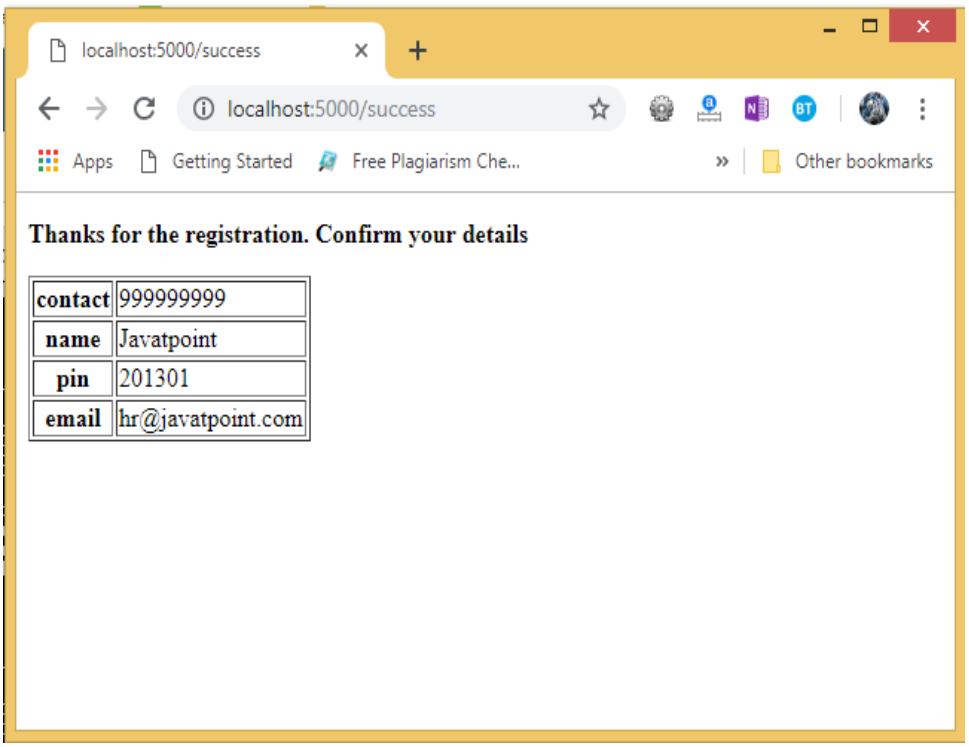
```
<html>
  <body>
    <h3>Register the customer, fill the following form.</h3>
    <form action = "http://localhost:5000/success" method = "POST">
      <p>Name <input type = "text" name = "name" /></p>
      <p>Email <input type = "email" name = "email" /></p>
      <p>Contact <input type = "text" name = "contact" /></p>
      <p>Pin code <input type ="text" name = "pin" /></p>
      <p><input type = "submit" value = "submit" /></p>
    </form>
  </body>
</html>
```

result_data.html

```
<!doctype html>
<html>
  <body>
    <p><strong>Thanks for the registration. Confirm your details</strong>
</p>
    <table border = 1>
      {% for key, value in result.items() %}
        <tr>
          <th> {{ key }} </th>
          <td> {{ value }} </td>
        </tr>
      {% endfor %}
    </table>
  </body>
```

14. </html>


localhost:5000
localhost:5000
Apps
Getting Started
Free Plagiarism Che...
Other bookmarks
Register the customer, fill the following form.
Name Javatpoint
Email hr@javatpoint.com
Contact 999999999
Pin code 201301
submit


localhost:5000/success
localhost:5000/success
Apps
Getting Started
Free Plagiarism Che...
Other bookmarks
Thanks for the registration. Confirm your details
contact 999999999
name Javatpoint
pin 201301
email hr@javatpoint.com

Flask Session

1. the session data is stored on the server.
2. The session can be defined as the duration for which a user logs into the server and logs out.
3. The data which is used to track this session is stored into the temporary directory on the server.
4. In the flask, a session object is used to track the session data which is a dictionary object that contains a key-value pair of the session variables and their associated values.
5. The following syntax is used to set the session variable to a specific value on the server.

   session[<variable-name>] = <value>

6. To remove a session variable, use the pop() method on the session object and mention the variable to be removed.

   session.pop(<variable-name>, none)