Deployment using Flask

Tushar B. Kute,
http://tusharkute.com

Web Framework

- Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

Flask

- Flask is a web application framework written in Python. It is developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco.
- Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.
- WSGI
  - Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

Flask

- WSGI
  - Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.
- Werkzeug
  - It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

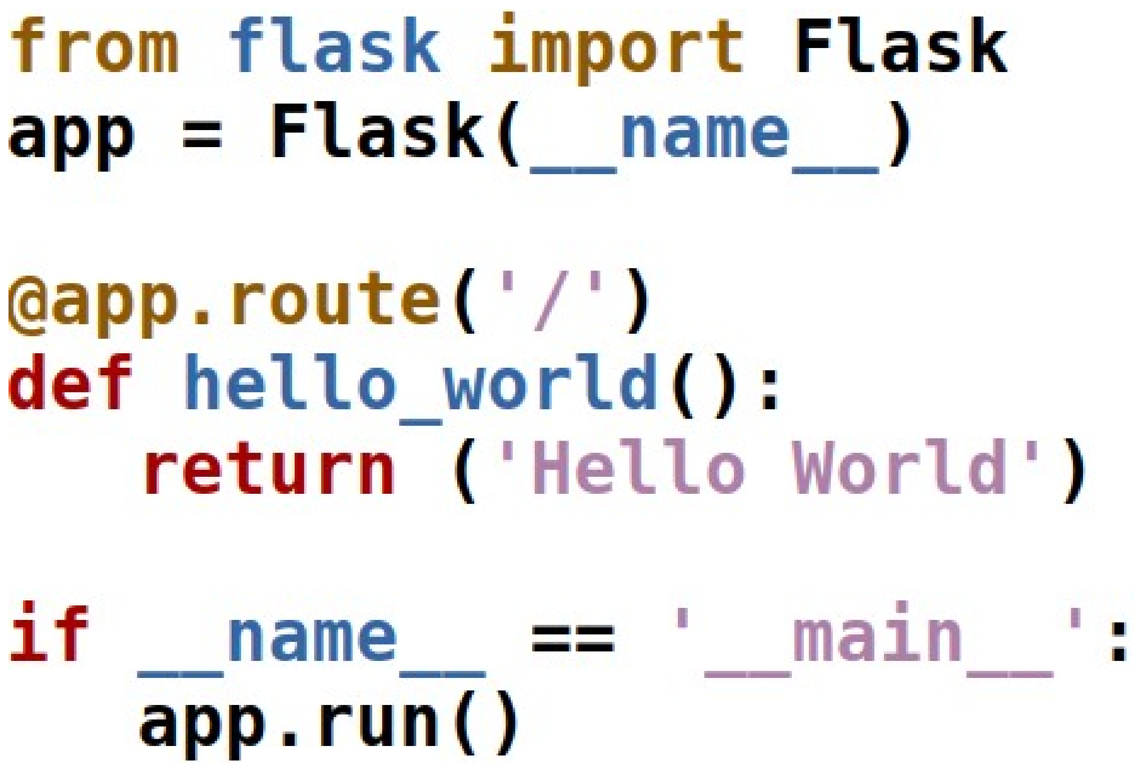
Jinja2

- Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.
- Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible.
- Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application.

Installation

- Install flask
  - `pip install flask`

hello.py



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

@app.route('/')
def hello_world():
    return ('Hello World')

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

hello.py

- Importing flask module in the project is mandatory. An object of Flask class is our WSGI application.
- Flask constructor takes the name of current module (__name__) as argument.
- The route() function of the Flask class is a decorator, which tells the application which URL should call the associated function.

Hello world

- `app.route(rule, options)`
  - The rule parameter represents URL binding with the function.
  - The options is a list of parameters to be forwarded to the underlying Rule object.
- In the above example, ‘/’ URL is bound with hello_world() function. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.
- Finally the run() method of Flask class runs the application on the local development server.
  - `app.run(host, port, debug, options)`

Parameters to app.run

- host
  - Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to ‘0.0.0.0’ to have server available externally
- port
  - Defaults to 5000
- debug
  - Defaults to false. If set to true, provides a debug information
- options
  - To be forwarded to underlying Werkzeug server.

Debug mode

- A Flask application is started by calling the run() method. However, while the application is under development, it should be restarted manually for each change in the code. To avoid this inconvenience, enable debug support. The server will then reload itself if the code changes. It will also provide a useful debugger to track the errors if any, in the application.
- The Debug mode is enabled by setting the debug property of the application object to True before running or passing the debug parameter to the run() method.

```
app.debug = True

app.run()

app.run(debug = True)
```

Routing

- Modern web frameworks use the routing technique to help a user remember application URLs. It is useful to access the desired page directly without having to navigate from the home page.
- The route() decorator in Flask is used to bind URL to a function. For example –

```
@app.route(‘/hello’)
def hello_world():
   return ‘hello world’
```

- Here, URL ‘/hello’ rule is bound to the hello_world() function. As a result, if a user visits http://localhost:5000/hello URL, the output of the hello_world() function will be rendered in the browser.

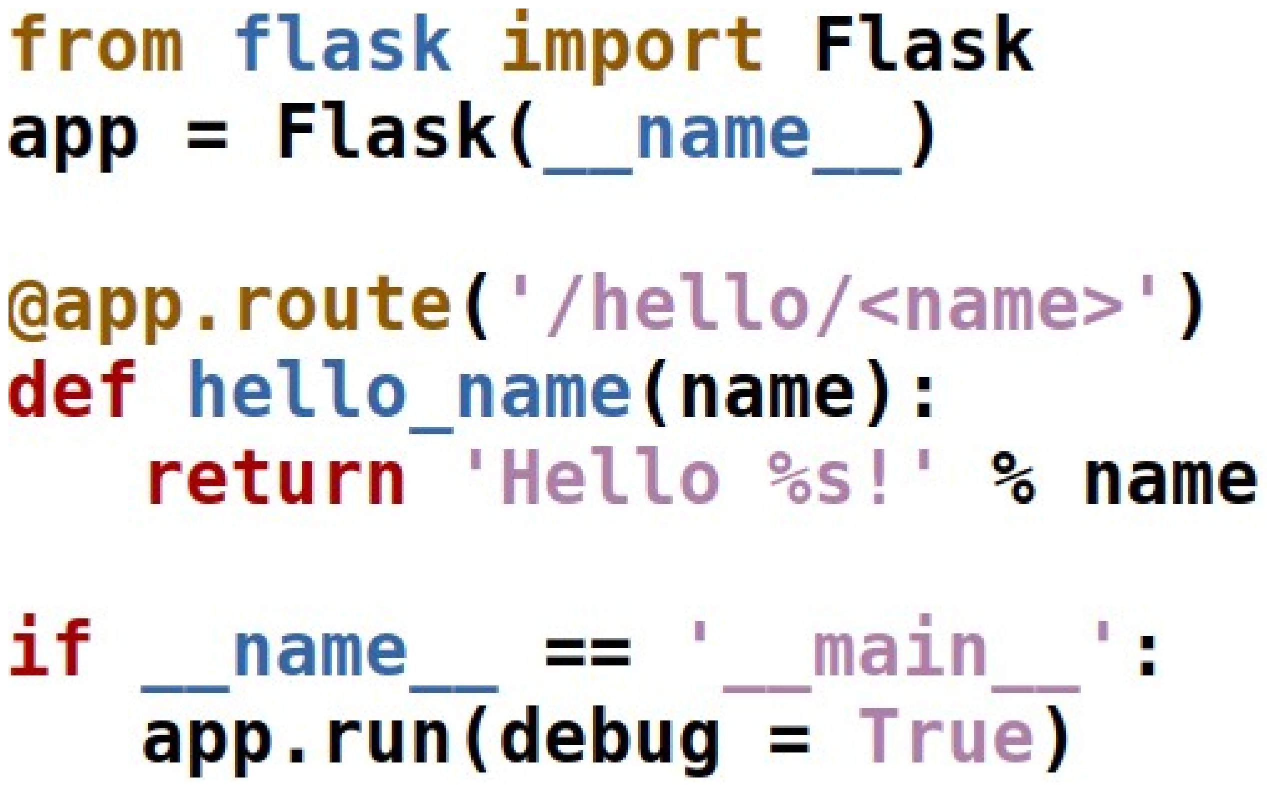
Example:

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

@app.route('/hello')
def hello_world():
    return ('Hello World')

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

Variable Types

- It is possible to build a URL dynamically, by adding variable parts to the rule parameter. This variable part is marked as <variable-name>.
- It is passed as a keyword argument to the function with which the rule is associated.
- In the following example, the rule parameter of route() decorator contains <name> variable part attached to URL '/hello'.
- Hence, if the http://localhost:5000/hello/Tushar is entered as a URL in the browser, 'Tushar' will be supplied to hello() function as argument.

Example:



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

@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello %s!' % name

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

Variables in path

- In addition to the default string variable part, rules can be constructed using the following converters –
- int
  - accepts integer
- float
  - for floating point value
- path
  - accepts slashes used as directory separator character

Example:

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

@app.route('/blog/<int:postID>')
def show_blog(postID):
   return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')
def revision(revNo):
   return 'Revision Number %f' % revNo

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

Output:

- After running the code from Python Shell. Visit the URL http://localhost:5000/blog/11 in the browser.
- The given number is used as argument to the show_blog() function. The browser displays the following output –

```
Blog Number 11
```

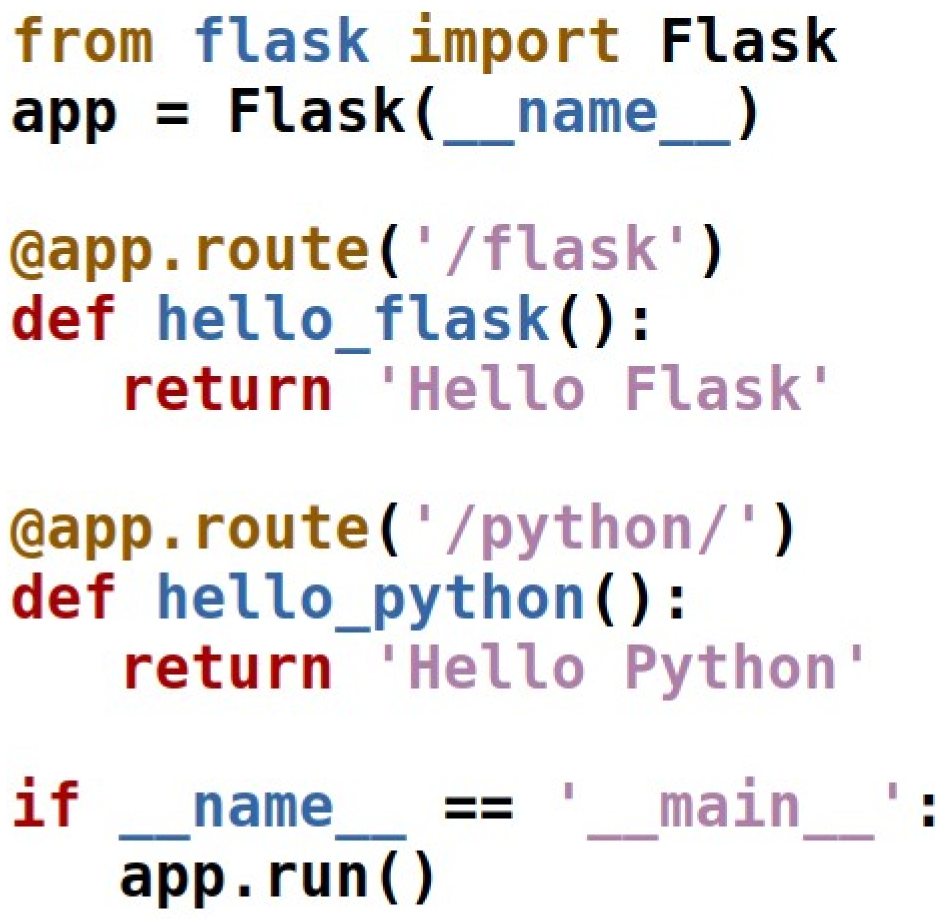
- Enter this URL in the browser – http://localhost:5000/rev/1.1
- The revision() function takes up the floating point number as argument. The following result appears in the browser window –

```
Revision Number 1.100000
```

Werkzeug’s Routing Module

- The URL rules of Flask are based on Werkzeug’s routing module.
- This ensures that the URLs formed are unique and based on precedents laid down by Apache.

Example:



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

@app.route('/flask')
def hello_flask():
    return 'Hello Flask'

@app.route('/python/')
def hello_python():
    return 'Hello Python'

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

Werkzeug’s Routing Module

- Both the rules appear similar but in the second rule, trailing slash (/) is used. As a result, it becomes a canonical URL.
- Hence, using /python or /python/ returns the same output. However, in case of the first rule, /flask/ URL results in 404 Not Found page.

URL Binding

- The url_for() function is very useful for dynamically building a URL for a specific function.
- The function accepts the name of a function as first argument, and one or more keyword arguments, each corresponding to the variable part of URL.

Example:

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def hello_admin():
   return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):
   return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
   if name =='admin':
      return redirect(url_for('hello_admin'))
   else:
      return redirect(url_for('hello_guest',guest = name))

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

Http Methods

- GET
  - Sends data in unencrypted form to the server. Most common method.
- HEAD
  - Same as GET, but without response body
- POST
  - Used to send HTML form data to server. Data received by POST method is not cached by server.
- PUT
  - Replaces all current representations of the target resource with the uploaded content.
- DELETE
  - Removes all current representations of the target resource given by a URL

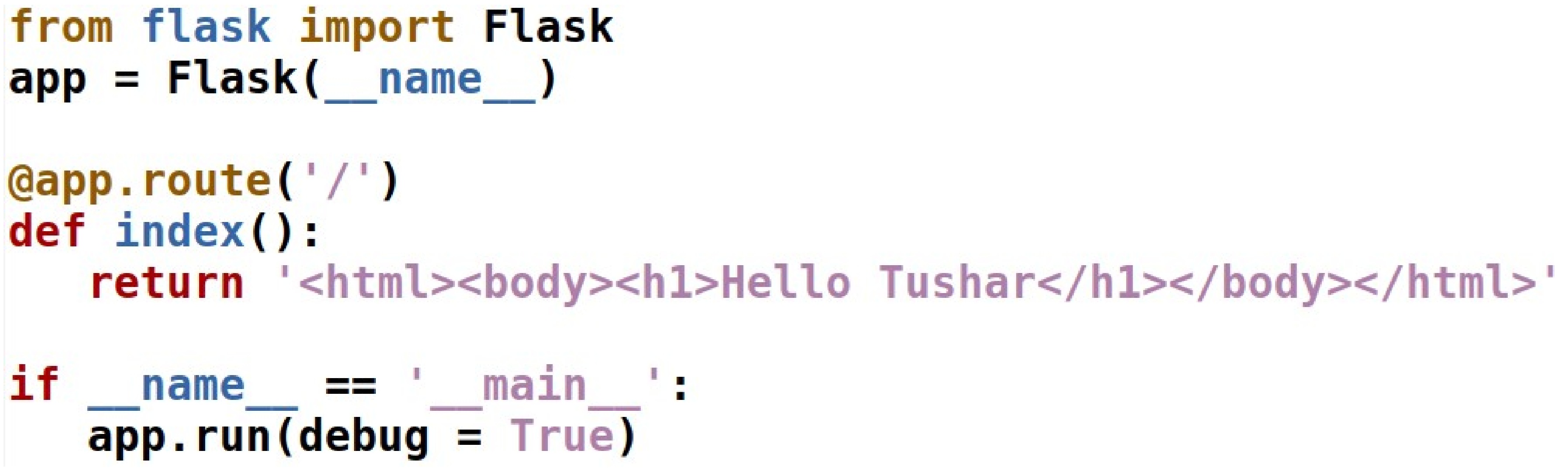
Html file

```html
<html>
   <body>
      <form action = "http://localhost:5000/login" method = "post">
         <p>Enter Name:</p>
         <p><input type = "text" name = "nm" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

Login.py

```python
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<name>')
def success(name):
   return 'welcome %s' % name

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST':
      user = request.form['nm']
      return redirect(url_for('success',name = user))
   else:
      user = request.args.get('nm')
      return redirect(url_for('success',name = user))

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

Render simple HTML



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

@app.route('/')
def index():
   return '<html><body><h1>Hello Tushar</h1></body></html>'

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

Templates

- Generating HTML content from Python code is cumbersome, especially when variable data and Python language elements like conditionals or loops need to be put. This would require frequent escaping from HTML.
- This is where one can take advantage of Jinja2 template engine, on which Flask is based. Instead of returning hardcode HTML from the function, a HTML file can be rendered by the render_template() function.

templates/helloworld.html

```html
<html>
    <head>
        <title>Basic Web Page</title>
    </head>
    <body>
        Hello World!
    </body>
</html>
```

Rendering template

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

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

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

Call template with data

- The term ‘web templating system’ refers to designing an HTML script in which the variable data can be inserted dynamically.
- A web template system comprises of a template engine, some kind of data source and a template processor.
- Flask uses jinja2 template engine. A web template contains HTML syntax interspersed placeholders for variables and expressions (in these case Python expressions) which are replaced values when the template is rendered.

Example:

```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<user>')
def index(user):
   return render_template('helloname.html', name=user)

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

Html with data

```
<html>
    <head>
        <title>Basic Web Page</title>
    </head>
    <body>
        Hello {{ name }}!
    </body>
</html>
```

Jinja2 delimiters

- The jinja2 template engine uses the following delimiters for escaping from HTML.
  - {% ... %} for Statements
  - {{ ... }} for Expressions to print to the template output
  - {# ... #} for Comments not included in the template output
  - # ... ## for Line Statements

Example:

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

@app.route('/hello/<int:score>')
def hello_name(score):
   return render_template('hello1.html', marks = score)

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

Html file

```
<html>
   <body>
      {% if marks>50 %}
         <h1> Your result is pass!</h1>
      {% else %}
         <h1>Your result is fail</h1>
      {% endif %}
   </body>
</html>
```

Using for loop

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

@app.route('/result')
def result():
   dict = {'phy':50,'che':60,'maths':70}
   return render_template('result1.html', result = dict)

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

Html file

```
<html>
   <body>
      <table border = 2>
         {% for key, value in result.items() %}
            <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>
         {% endfor %}
      </table>
   </body>
</html>
```

Static files

- A web application often requires a static file such as a javascript file or a CSS file supporting the display of a web page.
- Usually, the web server is configured to serve them for you, but during the development, these files are served from static folder in your package or next to your module and it will be available at /static on the application.
- A special endpoint ‘static’ is used to generate URL for static files.

Example:

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

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

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

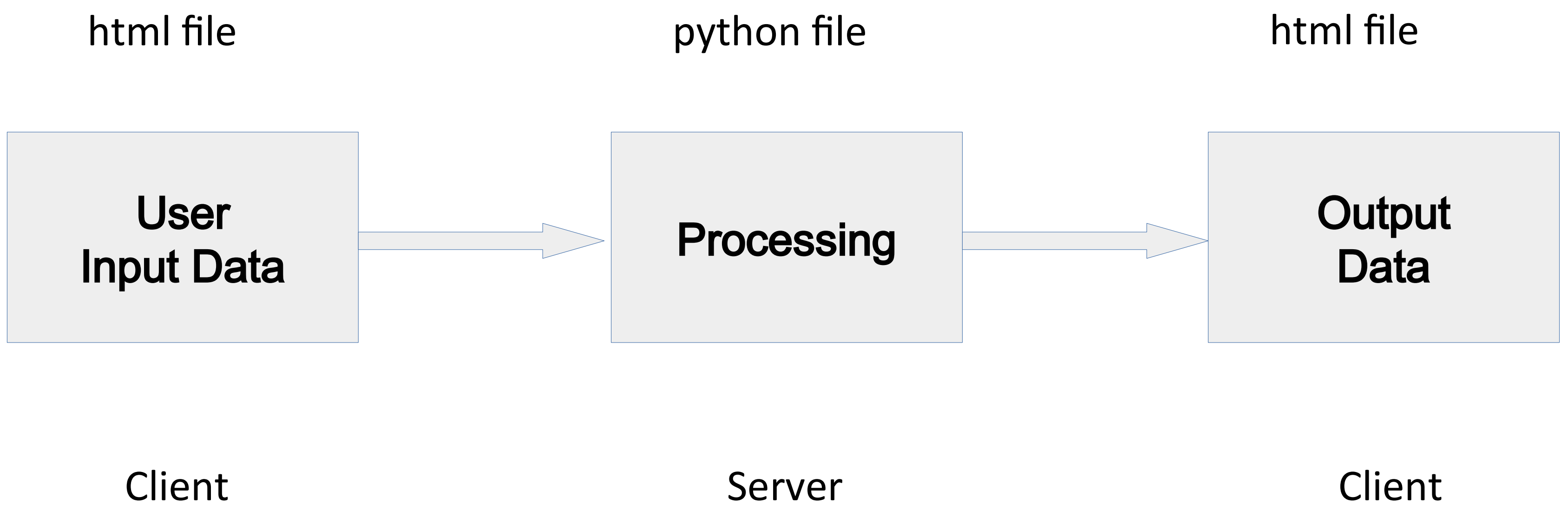
Html file

```
<html>
   <head>
      <script type = "text/javascript"
        src = "{{ url_for('static', filename = 'hello.js') }}" >
      </script>
   </head>

   <body>
      <input type = "button" onclick = "sayHello()" value = "SayHello" />
   </body>
</html>
```

static/hello.js file

```
function sayHello() {
   alert("Hello World")
}
```

Request object



Request object

- The data from a client’s web page is sent to the server as a global request object. In order to process the request data, it should be imported from the Flask module.
- Important attributes of request object are listed below –
  - Form – It is a dictionary object containing key and value pairs of form parameters and their values.
  - args – parsed contents of query string which is part of URL after question mark (?).
  - Cookies – dictionary object holding Cookie names and values.
  - files – data pertaining to uploaded file.
  - method – current request method.

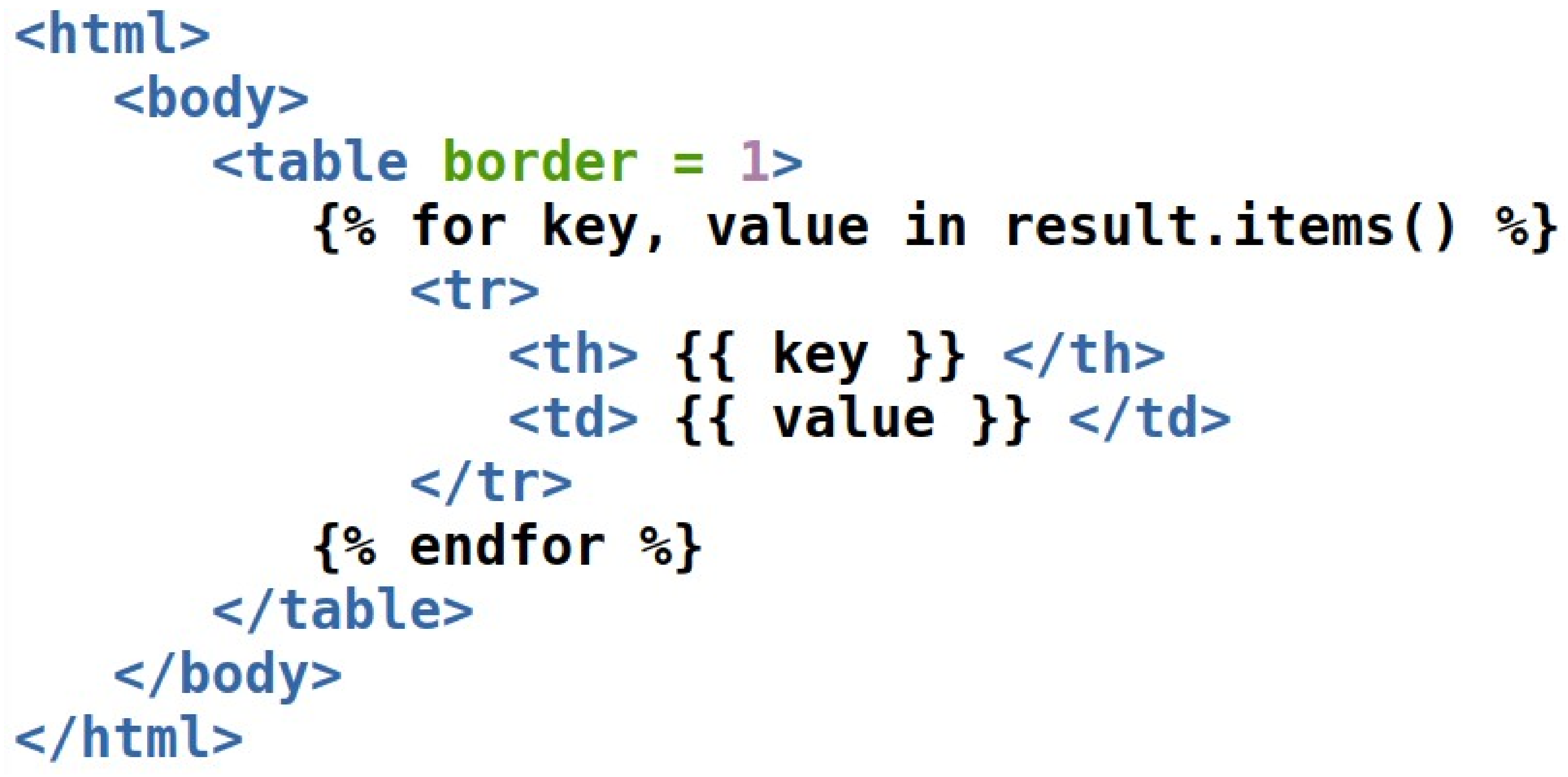
Main html file

```
<html>
  <body>
    <form action = "http://localhost:5000/result" method = "POST">
      <p>Name <input type = "text" name = "Name" /></p>
      <p>Physics <input type = "text" name = "Physics" /></p>
      <p>Chemistry <input type = "text" name = "chemistry" /></p>
      <p>Maths <input type ="text" name = "Mathematics" /></p>
      <p><input type = "submit" value = "submit" /></p>
    </form>
  </body>
</html>
```

Flask code

```python
from flask import Flask, render_template, request
app = Flask(__name__)

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

@app.route('/result',methods = ['POST', 'GET'])
def result():
   if request.method == 'POST':
      result = request.form
      return render_template("result2.html",result = result)

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

Result



```
<html>
   <body>
      <table border = 1>
         {% for key, value in result.items() %}
            <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>
         {% endfor %}
      </table>
   </body>
</html>
```

Cookie

- A cookie is stored on a client’s computer in the form of a text file.
- Its purpose is to remember and track data pertaining to a client’s usage for better visitor experience and site statistics.
- A Request object contains a cookie’s attribute.
- It is a dictionary object of all the cookie variables and their corresponding values, a client has transmitted. In addition to it, a cookie also stores its expiry time, path and domain name of the site.

Cookie

- In Flask, cookies are set on response object. Use make_response() function to get response object from return value of a view function.
- After that, use the set_cookie() function of response object to store a cookie.
- Reading back a cookie is easy.
  - The get() method of request.cookies attribute is used to read a cookie.

Sample login form application

```python
@app.route('/success',methods = ['POST'])
def success():
    if request.method == "POST":
        email = request.form['email']
        password = request.form['pass']

    if password=="admin123":
        resp = make_response(render_template('success.html'))
        resp.set_cookie('email',email)
        return resp
    else:
        return redirect(url_for('error'))

@app.route('/viewprofile')
def profile():
    email = request.cookies.get('email')
    resp = make_response(render_template('profile.html',name = email))
    return resp
```

Session

- Like Cookie, Session data is stored on client. Session is the time interval when a client logs into a server and logs out of it. The data, which is needed to be held across this session, is stored in the client browser.
- A session with each client is assigned a Session ID. The Session data is stored on top of cookies and the server signs them cryptographically. For this encryption, a Flask application needs a defined SECRET_KEY.
- Session object is also a dictionary object containing key-value pairs of session variables and associated values.

Creating Session

- Session object is also a dictionary object containing key-value pairs of session variables and associated values.
- For example, to set a ‘username’ session variable use the statement –

```
Session[‘username’] = ’admin’
```

- To release a session variable use pop() method.

```
session.pop('username', None)
```

Sample Session

```python
@app.route('/success',methods = ["POST"])
def success():
    if request.method == "POST":
        session['email']=request.form['email']
        return render_template('success.html')

@app.route('/logout')
def logout():
    if 'email' in session:
        session.pop('email',None)
        return render_template('logout.html');
    else:
        return '<p>user already logged out</p>'

@app.route('/viewprofile')
def profile():
    if 'email' in session:
        email = session['email']
        return  render_template('profile.html',name=email)
```

Redirect and Errors

- Flask class has a redirect() function. When called, it returns a response object and redirects the user to another target location with specified status code.
- Prototype of redirect() function is as below –

```
Flask.redirect(location, statuscode, response)
```

- In the above function –
  - location parameter is the URL where response should be redirected.
  - statuscode sent to browser's header, defaults to 302.
  - response parameter is used to instantiate response.

Status Codes

- The following status codes are standardized –
  - HTTP_300_MULTIPLE_CHOICES
  - HTTP_301_MOVED_PERMANENTLY
  - HTTP_302_FOUND
  - HTTP_303_SEE_OTHER
  - HTTP_304_NOT_MODIFIED
  - HTTP_305_USE_PROXY
  - HTTP_306_RESERVED
  - HTTP_307_TEMPORARY_REDIRECT
- The default status code is 302, which is for ‘found’.

Sample Login Page

```
<html>
  <head>
    <title>Login Page</title>
  </head>
  <body>
    <form method = "post" action = "http://localhost:5000/login">
      <table>
        <tr><td>Username</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" value = "Submit"></td></tr>
      </table>
    </form>
  </body>
</html>
```

Python Code

```
@app.route('/')
def index():
   return render_template('log_in.html')

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST' and request.form['uname'] == 'admin' :
      return redirect(url_for('success'))
   else:
      return redirect(url_for('index'))

@app.route('/success')
def success():
   return 'logged in successfully'
```

Abort

- Flask class has abort() function with an error code.

  `Flask.abort(code)`
- The Code parameter takes one of following values –
  - 400 – for Bad Request
  - 401 – for Unauthenticated
  - 403 – for Forbidden
  - 404 – for Not Found
  - 406 – for Not Acceptable
  - 415 – for Unsupported Media Type
  - 429 – Too Many Requests

Example:

```
@app.route('/')
def index():
   return render_template('log_in.html')

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST':
      if request.form['uname'] == 'admin' :
         return redirect(url_for('success'))
      else:
         abort(400)
   else:
      return redirect(url_for('index'))

@app.route('/success')
def success():
   return 'logged in successfully'
```

File Uploading

- Handling file upload in Flask is very easy. It needs an HTML form with its enctype attribute set to ‘multipart/form-data’, posting the file to a URL.
- The URL handler fetches file from request.files[] object and saves it to the desired location.
- Each uploaded file is first saved in a temporary location on the server, before it is actually saved to its ultimate location.
- Name of destination file can be hard-coded or can be obtained from filename property of request.files[file] object.
- However, it is recommended to obtain a secure version of it using the secure_filename() function.

App configuration

- It is possible to define the path of default upload folder and maximum size of uploaded file in configuration settings of Flask object.
- app.config[‘UPLOAD_FOLDER’]
  - Defines path for upload folder
- app.config[‘MAX_CONTENT_PATH’]
  - Specifies maximum size of file yo be uploaded – in bytes

Upload html file

```
<html>
  <body>
    <form action = "http://localhost:5000/uploader" method = "POST"
      enctype = "multipart/form-data">
      <input type = "file" name = "file" />
      <input type = "submit"/>
    </form>
  </body>
</html>
```

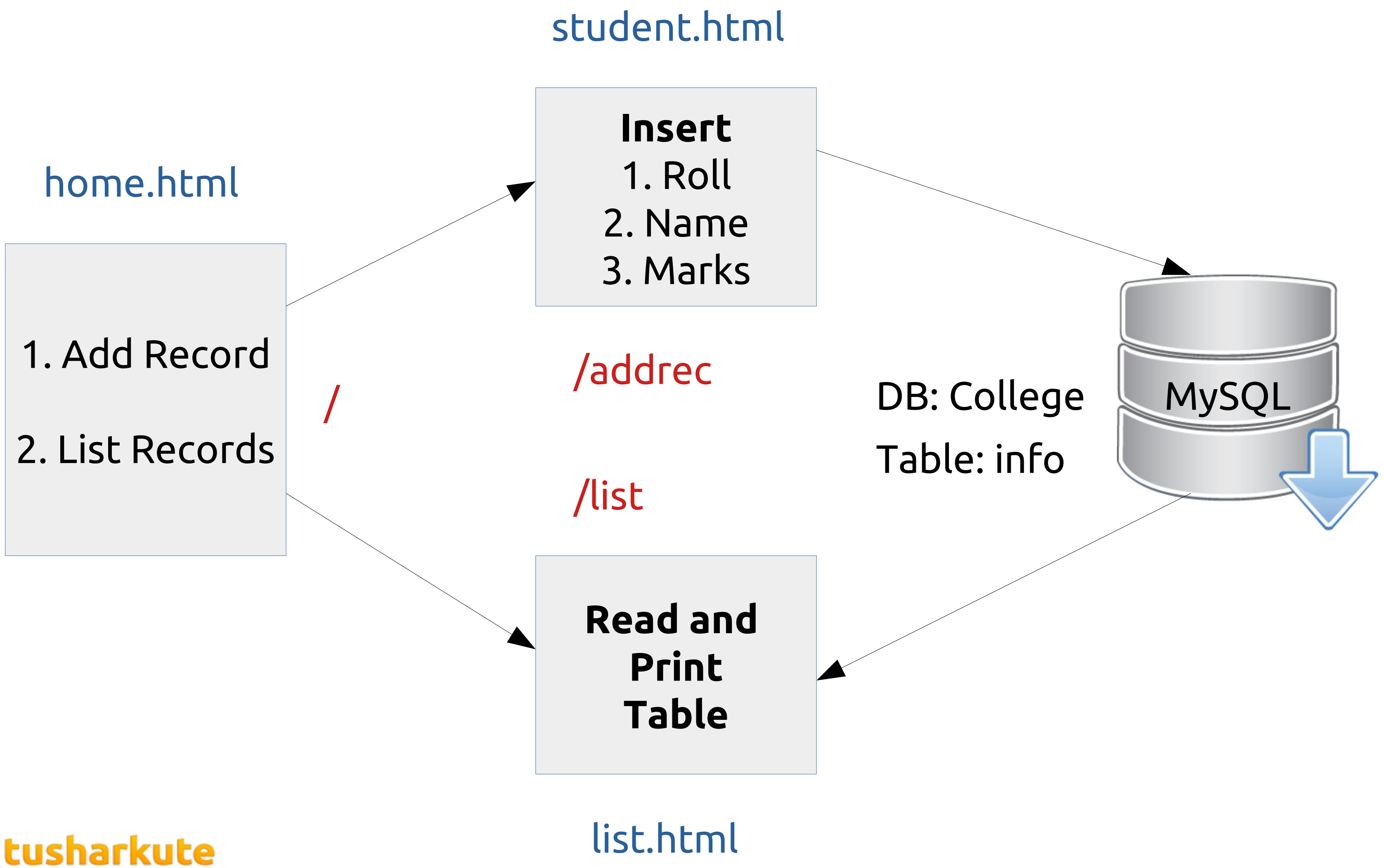
Python code

```
from flask import Flask, render_template, request
from werkzeug.utils import secure_filename
app = Flask(__name__)

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

@app.route('/uploader', methods = ['GET', 'POST'])
def uploader():
   if request.method == 'POST':
      f = request.files['file']
      f.save(secure_filename(f.filename))
      return 'file uploaded successfully'

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

Database Connectivity

- Front end + Back end application
- Common Databases:
  - Mysql
  - Sqlite
  - MS SQL Server
  - Oracle Server

The Database Connection



Home page

```html
<html>
    <head>
        <title>My Home Page</title>
    </head>
    <body>
          <h1>Welcome to the website</h1>
          <h3><a href = "/student">Add New Record</a></h3>
          <h3><a href = "/list">View All Records</a></h3>
    </body>
</html>
```

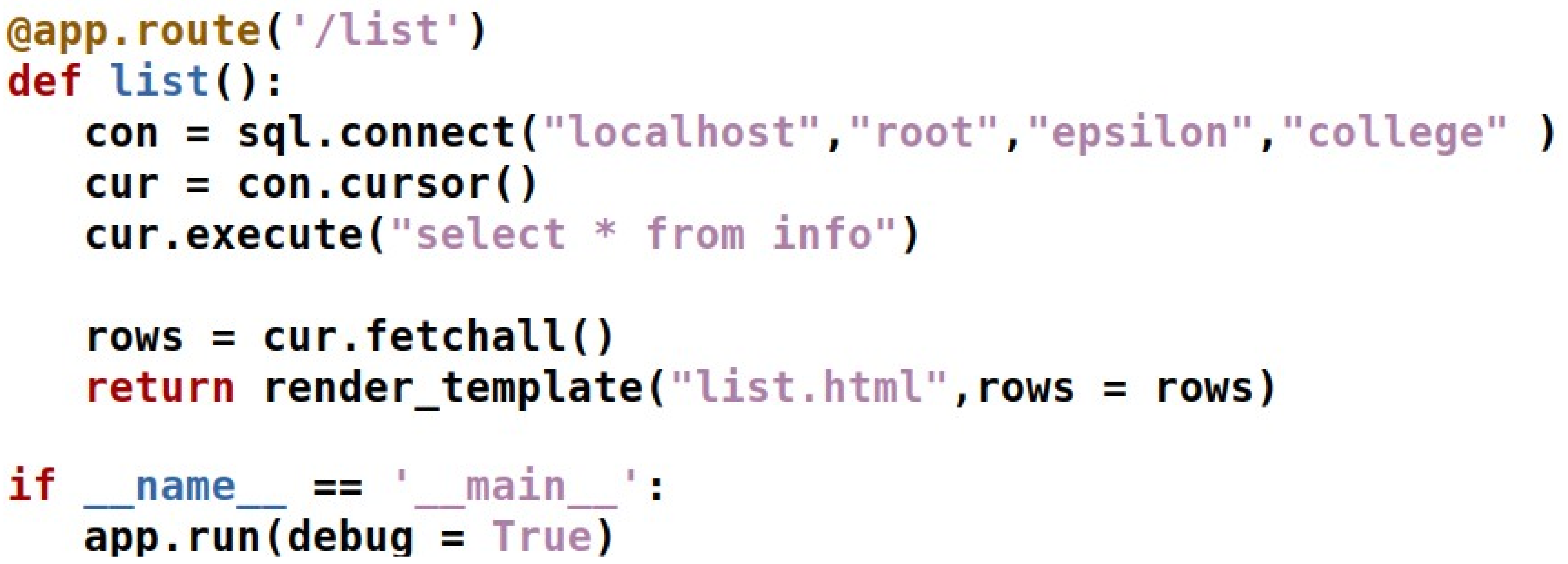
Python Code

```
from flask import Flask, redirect, url_for,
request, render_template
import MySQLdb as sql
app = Flask(__name__)

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

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

Python Code

```python
@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   if request.method == 'POST':
      try:
         name = request.form['name']
         roll = int(request.form['roll'])
         marks = float(request.form['marks'])

         con = sql.connect("localhost","root","epsilon","college" )
         cur = con.cursor()
         query = 'insert into info values (%d,"%s",%f)' %(roll, name, marks)
         cur.execute(query)

         con.commit()
         msg = "Record successfully added"
         con.close()
      except Exception as e:
         con.rollback()
         msg = "Error in insert operation" + e
         con.close()
      finally:
         return render_template("result.html", msg = msg)
```

Python Code



```python
@app.route('/list')
def list():
   con = sql.connect("localhost","root","epsilon","college" )
   cur = con.cursor()
   cur.execute("select * from info")

   rows = cur.fetchall()
   return render_template("list.html",rows = rows)

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

Student.html

```
<html>
   <body>
      <form action = "http://localhost:5000/addrec" method = "POST">
       <h3>Student Information</h3>
         Roll<br>
         <input type = "text" name = "roll" /><br>
         Name<br>
         <input type = "text" name = "name" /></br>
         Marks<br>
         <input type = "text" name = "marks" /><br>
         <input type = "submit" value = "submit" /><br>
      </form>
   </body>
</html>
```

List.html

```
<html>
   <body>
      <table border = 1>
         <thead>
            <td>Roll</td>
            <td>Name</td>
            <td>Marks</td>
         </thead>
         {% for row in rows %}
            <tr>
               <td>{{row[0]}}</td>
               <td>{{row[1]}}</td>
               <td>{{row[2]}}</td>
            </tr>
         {% endfor %}
      </table>
      <a href = "/">Go back to home page</a>
   </body>
</html>
```

Thank you

@mitu_skillologies @mITuSkillologies @mitu_group @mitu-skillologies @MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com