LECTURE 13 WEB APPLICATION DEVELOPMENT WITH FLASK

What is Flask?

- Flask is a web application framework written in Python
  - 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.
  - can be used to create front end (HTML5, CSS, jQuery or Javascript
- Created by an international Python community called Pocco, based on 2 other projects:
  - Werkzeug: A Web Server Gateway Interface (WSGI) that handles HTML requests, responses, etc.
  - Jinja2: a web templating system combining a template with data sources to render dynamic web pages.

Adapted from Karen Works's COP4521 material, which in turn was adapted from:
_https://www.tutorialspoint.com/flask/flask_overview.htm

Installation

- Not in a typical Python installation
- See the official document: https://flask.palletsprojects.com/en/3.0.x/
- Flask has been installed on linprog

FLASK application: ‘Hello World’

```
# Starting a web page at http://127.0.0.1:50000/

from flask import Flask
app = Flask(__name__)

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

if __name__ == '__main__':
   app.run(host=‘127.0.0.1’, port = 50000)
```

Import the Flask class, an instance
Of the class is the web application

Create the class instance

Route() decorator tells Flask the
hello_world() function will be triggered
When ‘/’ in the website is accessed, the
Web application will return the return value
Of hello_world() to the web browser

Start the website at port 50000 on the
Local machine (127.0.0.1)

- After ‘python3 FlaskHelloWorld.py linprog3’ on linprog3. We can telnet do ‘telnet linprog3 50000’ and then issue ‘GET / HTTP/1.0\r\n\r\n’ to see the ‘Hello World’ response from the Flask website

Flask – Application

- **The run() method** of Flask class runs the application on the local development server.
- app.run(host, port, debug, options)
  - Note : All parameters are optional
  - host: Hostname to listen on. Defaults to 127.0.0.1 (localhost).
  - port: defaults to 5000
  - debug: defaults to false. If set to true, provides a debug information
  - options: to be forwarded to underlying (Werkzeug) server.
- Example: app.run(host='127.0.0.1', port = 50000) starts a website at http://127.0.0.1:50000

Flask – 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)
```

Flask – Application

- **The route() function** tells the application which URL in the website should call the associated function.
- app.route(rule, options) binds URL to a function.
- 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 example, with app.run(host='127.0.0.1', port = 50000) (website at http://127.0.0.1:50000). @app.route('/') says that URL 'http://127.0.0.1:50000/' 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.

Flask – more example

- 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.

```
# starting a web page at http://localhost:5000/hello
from flask import Flask
app = Flask(__name__)

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

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

Flask – Routing

- The add_url_rule() function of an application object is also available to bind a URL with a function as in the above example, route() is used.
  - First parameter is the rule, the third parameter is the function. The second parameter is called engpoint, which is the name of the route.

```
def hello_world():
    return 'Hello World'
app.add_url_rule('/hello', 'hello_world', hello_world)
```

Flask – Variable Rules in routing

- Flask **variable rules** allow for the creation of a dynamic URL by adding variable parts, to the rule parameter.
  - Dynamic URL: 'http://localhost:5000/guest/{guest_name}'
    - ❖ {guest_name} is a place holder for any guest name
- A variable rule can be defined by <variable-name> in the rule such as '/guest/<name>'.
- The variable must be passed as an argument to the function with which the rule is associated in Flask

Flask – Variable Rules

- 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/Test is entered as a URL in the browser, 'Test' will be supplied to hello() function as argument.
- See FlaskURLBuilding.py

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

Flask – Variable Rules

- There can be more than one variables in a rule as shown in the following example. All variables must be passed to the associated function

```
@app.route('/posts/<post_id>/<slug>’)
def show_post(post_id, slug):
    return f"Post {post_id} - Slug: {slug}"
```

Flask – URL Building

- Once we define the URL rules (patterns) and associate them with a function, we can reuse them in code and the templates.
- url_for() function is 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.

```
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)
```

Flask – HTTP methods

Sr.No.	Methods & Description
1	**GET** Sends data in unencrypted form to the server. Most common method.
2	**HEAD** Same as GET, but without response body
3	**POST** Used to send HTML form data to server. Data received by POST method is not cached by server.
4	**PUT** Replaces all current representations of the target resource with the uploaded content.
5	**DELETE** Removes all current representations of the target resource given by a URL

```
GET /contact HTTP/1.1
Host: example.com
User-Agent: curl/8.6.0
Accept: */*
```

```
POST /test HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

field1=value1&field2=value2
```

Flask – HTTP methods

- By default, the Flask route responds to the GET requests.

- This can be altered by providing methods argument to route() decorator.

```
@app.route('/posts', methods=['GET', 'POST'])
def posts():
  if request.methods == 'GET':
        return 'get the posts'
  elif request.methods == 'POST':
        return 'make a new post'
```

For form submission, how to access the values in the form?

- Let us review what we learn so far:
- Modify the following Flask code to run Flask web pages at http://www.cs.fsu.edu:10000/cop4521 and http://www.cs.fsu.edu:10000/cop4610

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

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

if __name__ == '__main__':
   app.run(host=‘127.0.0.1’, port = 50000)
```

- Modify the following Flask code to run dynamic Flask websites that allow each student to have a separate web page at http://www.cs.fsu.edu:10000/cop4521/<student_id>

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

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

if __name__ == '__main__':
   app.run(host=‘127.0.0.1’, port = 50000)
```

- Modify the following Flask code to run a Flask website that can respond to http GET and PUT methods, http://www.cs.fsu.edu:10000/cop4521

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

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

if __name__ == '__main__':
   app.run(host=‘127.0.0.1’, port = 50000)
```

What we know so far

- We can associate a URL with a python function.
  - The return value of the function will be sent back when a user access the URL.
- We can associate dynamic URLs with a function
- We can reuse the code with url_for() and redirect()
- We can respond to different HTTP requests (GET, POST, etc)

- Since Flask is intended to be used to build websites, the response needs to be in HTML! This is where Flask templates come into play!

Flask – HELLO WORLD in HTML

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

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

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

Flask – Rendering HTML response using render_template()

- A HTML response can be rendered by the **render_template()** function.
- Render_template() takes an HTML template file as an argument
  - The HTML template file should be put in the subdirectory templates under the python source code.
- Application file structure:
  - Application folder
    - Hello.py
    - templates
      - hello.html
- See FlaskRenderTemplateVer1

```python
from flask import Flask, render_template

app = Flask(__name__)

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

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

Flask – Passing arguments to the HTML template

- Render_template() takes any number of keyword arguments.
- Accessing arguments in the HTML template with {{ argument name }}
- See FlaskRenderTemplateVer2

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

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

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

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Hello {{ name }}!</h1>
</body>
</html>
```

Passing argument into the HTML template

Accessing the argument in the HTML template

Flask – More escapes from HTML in the HTML template.

- 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
- See FlaskRenderTemplateVer3

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

Flask – Dynamic content in the HTML response

- Display a dynamic table (key-value pair)
  - For loop are enclosed in {%..%} whereas, the expressions key and value are put inside {{ }}.
- See FlaskRenderTemplateVer4

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

Flask – More dynamic content in the HTML response

- See FlaskRenderTemplateVer5: Display a dynamic table

```
<!doctype html>
<html>
  <body>
    <table border = 1>
      {% for row in result) %}
        <tr>
          {% for item in row %}
          <td> {{ value }} </td>
          {% endfor %}
        </tr>
      {% endfor %}
    </table>
  </body>
</html>
```

Flask – Handle forms and interact with sqlite3

- See ExampleCrapsSimpleFlaskWebsite

```
@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   if request.method == 'POST':
      try:
         nm = request.form['Name']
         ag = request.form['Age']
         amt = request.form['Money']
         pwd = request.form['Password']
     ….
     Put data in a sqlite3 database using SQL.
     '''
```

```
<body>
   …
  <form action="{{ url_for('addrec') }}"  method="POST">
     <h3>Player Information</h3>
     Name<br>
     <input type="text" name="Name" ><br>
     Age<br>
     <input type="text" name="Age"><br>
     Money<br>
     <input type="text" name="Money"><br>
     Password<br>
     <input type="password" name="Password"><br>
     <input type="submit" value="submit"><br>
  </form>
</body>
```