HOW TO FLASK

And a *very short* intro to web development and databases

FLASK

- Flask is a web application framework written in Python.
- Created by an international Python community called Pocco.
- Based on 2 other Python projects:
  - Werkzeug
    - It is a Web Server Gateway Interface (WSGI) toolkit that handles HTML requests, responses, etc.
  - Jinja2
    - A web templating system combines a template with a certain data source to render dynamic web pages.

INSTALLING FLASK

- Flask can be installed through pip
  - pip install Flask
- You can also install it inside a virtual environment (recommended)
- This will install several dependencies, including Werkzeug and jinja
- If you intend to use Flask with a SQL database, Python comes with SQLite3, but you might need to install some sub-libraries like flask-wtf and sqlalchemy

FLASK APPLICATION

- The flask constructor takes the name of the current module as the parameter.
- The route function of the Flask class is a decorator, which tells the application which URL should call the associated function.
- In the example, '/' URL is bound with hello_world. Hence, when the home page of web server is opened in a browser, the output of this function will be rendered.
- Finally the run method of the Flask class runs the application on the local development server.

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

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

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

FLASK APPLICATION

- The route function:
  - Signature: 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.
- The run function:
  - Signature: app.run(host, port, debug, options)
  - Hostname defaults to localhost (127.0.0.1)
  - Port number defaults to 5000.

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def hello_world():

  return 'Hello World'


if __name__ == '__main__':

  app.run()
```

FLASK-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.
- 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.

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

FLASK-VARIABLES

- 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/Spongebob is entered as a URL in the browser, 'Spongebob' will be supplied to hello_name() function as argument.

```
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)
```

HTTP REQUESTS AND RESPONSES

- When a client (browser) requests a URL, it sends a HTTP REQUEST to the server.
- The server then sends the resulting rendered HTML page as a HTTP RESPONSE.
- It is possible to send form information to the server as a part of the HTTP REQUEST.
- There are several ways to make a HTTP REQUEST. The most common ways are:
  - GET – Form information is a part of the URL. Not very secure. Amount of information is limited. Preferred for testing.
  - POST – Form information is not a part of the URL. Hidden from the world. Secure. No limit on the amount of information. Preferred for production environments.

TEMPLATES

- A flask application renders an HTML page upon requesting a URL.
- However, the files may not be static HTML. Sometimes, we want to add information from the request/URL into the resulting HTML.
- In order to do that, we have to incorporate placeholders into the HTML files, and place the files themselves into a folder called “templates”

```
<html>
        <body>
                <h1>Hello {{ name }} </h1>
        </body>
</html>
```

Placeholder

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 is where one can take advantage of **Jinja2** template engine
- Instead of returning hardcode HTML from the function, a HTML file can be rendered by the render_template() function.
- hello.html is the file on the previous slide.

```
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)
```

MESSAGE FLASHING

- It is important to give the user some feedback when something goes wrong.
- It is relatively easy on a console application – we can just raise an exception and print to the standard error stream.
- It is harder to do so on a client server GUI application.
- Generating such informative messages is easy in Flask web application.
- The Flask module contains the **flash()** method. It passes a message to the next request, which generally is a template.
  - Signature: flash(message, category)
  - **message** is the actual message to be flashed.
  - **category** is optional. It can be either ‘error’, ‘info’ or ‘warning’.

MESSAGE FLASHING

- In order to remove message from session, template calls **get_flashed_messages()**.
- The HTML template file would contain the following lines. They will be rendered by flash on the server side if an error occurs.

```
{% with messages = get_flashed_messages() %}
        {% if messages %}
                {% for message in messages %}
                        {{ message }}
                {% endfor %}
        {% endif %}
{% endwith %}
```

MESSAGE FLASHING

- The flash app would contain the following lines upon redirect to ‘login.html’.
- When submitted, the **login()** view function verifies a username and password and accordingly flashes a **‘success’** message or creates **‘error’** variable.

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
   error = None
      if request.method == 'POST':
      if request.form['username'] != 'admin' or  request.form['password'] != 'admin':
         error = 'Invalid username or password. Please try again!'
      else:
         flash('You were successfully logged in')
         return redirect(url_for('index'))
   return render_template('login.html', error = error)
```

SQLITE3

- sqlite is a database management software that is available with a python installation.
- The recommended version is sqlite3
- You an create a sqlite database on the terminal.
  - $> squite3 database.db
  - This creates a sqlite database called database.db
- Once the sqlite prompt opens, exit immediately.

SQLITE 3

- The setup.py file sets up a table in the database using python. This file does not use flask. Just sqlite.
- We first establish a connection to the database.
- We then execute a SQL query to create a table.
- Finally, we close the connection.

```
import sqlite3

conn = sqlite3.connect('database.db')

print ("Opened database successfully")


conn.execute('CREATE TABLE students (Name
      TEXT, Classes TEXT, Major TEXT, GPA TEXT)')

print ("Table created successfully")

conn.close()
```

USING SQL WITH FLASK

- Form data sent to the flask app is available in a flask variable called request.form.
- This is a dictionary. The key is the name attribute of the HTML form that was submitted.
- Connect to the database using sql.connect
- Grab the "cursor" of the database. This is like a file pointer.
- Execute the query using the cursor object.
- It is recommended that we commit the changes to the database if the query is successful and rollback the changes if the query failed.
- If the query were a select query, we can get the query results using the fetchall() function.
- Once we are done, close the connection.
- The results can be used to render various template HTML files.