COSC 419 – Topics in Computer Science

Fall 2020

Recap: New Flask Features

- Last lecture, we learnt about a number of features built into the Flask framework, including:
    - How to catch POST requests, and how to read data from a POST request using `request.form`
    - How to store, read, and delete data in a user-specific session using our Flask secret key
    - How we can “nest” together templates using the `block` and `extends` template commands
- We’re going to use all of these features in this week’s lab

Lab 2 Note and Quiz Announcement

- A note that I should have mentioned in the Lab 2 document: Not all URLs will work with your scraping script, and that’s not a problem!
  - Actually has nothing to do with our script, but rather that some websites will block typical “bot” user-agents, which includes the default user agent in Python
- I’m going to release quiz 1 this week – the quiz will be posted on Friday, and available for a period of 1 week
  - Should take about 20-30 minutes to complete

What We’re Doing Today

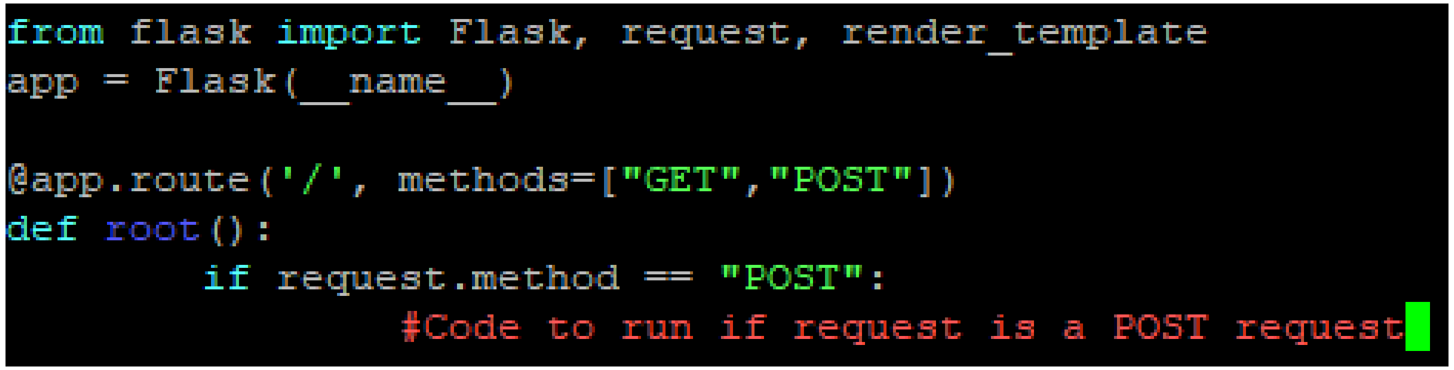
- Today, we’re going to be doing some demos of the techniques that we looked at last lecture
- We’re going to use some of the approaches demoed today in our lab this week, so it would be good to follow along
    - If you do follow along, please take a copy of your `myapp.py` file ***beforehand*** and name it to something like lab2_myapp.py so that I can mark your Lab 2
- How can we practically make use of these features to help us in designing a full stack application?

Technique #1: Get and Post in the same Route

- Since we can specify both GET and POST methods in a route, and also use `request.method` to check how a request was sent, we can use a single route to manage both GET and POST requests
- This is handy because it allows us to serve a form page when a GET request is received, and then have that form return a POST request to the same page
  - All logic related to both serving the form and handling the form request can be kept in a single function

Technique #1: Get and Post in the same Route

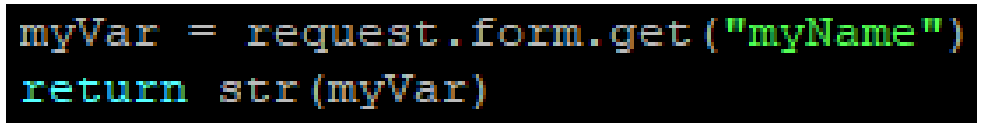
- Recall that `request.method` contains a string with the current HTTP request type:



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

@app.route('/', methods=["GET","POST"])
def root():
        if request.method == "POST":
                #Code to run if request is a POST request
```

- Also recall that we can use request.form to fetch POST request data just like we use request.args to fetch GET request query string parameters



```
myVar = request.form.get("myName")
return str(myVar)
```

Technique #2: Storing and Clearing Session Vars

- The second technique I want to demonstrate is setting and unsetting session variables
- A fairly simple test: We define one route that sets a session variable, one route that checks to see if the session variable exists and if so, print it out, and one route to clear the session variable
- We'll also look at what happens if we try to store a ***lot*** of data in the session, and why that doesn't really work.

Technique #2: Storing and Clearing Session Vars

- Recall that the first things we need to do to use a session is to import the `session` module from flask, and set our `app.secret_key` which will encrypt our session cookie
- After that, we can simply use the dictionary reference to assign a new session variable (as a key-value pair):

```
session[“myKey”] = “myValue”
```

- We can then use `session.get()` and `session.pop()` to read and remove the set session variable, respectively

Technique #3: Nesting Templates

- The last technique I want to demonstrate is how we can "nest" templates together to render HTML in chunks
- This has, historically, been one of the trickier concepts for students to grasp – specifically, the ordering of which template extends which template, and how multiple inheritance works
- Remember: the `render_template` call is made to the child, which `extends` it's parent template

Technique #3: Nesting Templates

- Besides `extends`, there's actually another keyword we can use in Jinja2 templates: `include`
- While the `extends` keyword only inserts HTML that is in matching named `blocks` between both the child and the parent, the `include` keyword will include all the HTML inside a file
- Handy for rendering elements on only some pages, as well as allowing you to keep your HTML "chunked" in files for easy maintenance (rather than one long HTML file)

This Week’s Lab

- In the lab today, we’ll be building a basic log-in web application which is going to use all three of these techniques:
  - We’ll serve a login form via GET and it will return a POST request with user login data
  - We’ll use a session to store whether the user is logged in or not, and prevent access to a “secure page” if they aren’t
  - And we’ll render the whole thing using some inherited templates to minimize how much HTML we’ll need

Any Questions?