Flask

Flask

Python's built-in web server is nice, but serious web development is done using a web framework. Web frameworks typically provide:

- Routes, which map URLs to server files or Python code
- Templates, which dynamically insert server-side data into pages of HTML
- Authentication and authorization of user names, passwords, permissions
- Sessions, which keep track of a user during a single visit to a site
- and more . . .

We'll use a simple Python web framework called Flask.

Installing Flask

To install Flask, use pip:

```
$ pip3 install flask
```

To check that your Flask installation was successful, import it:

```
>>> import flask
```

If you get no error messages, you're ready to start developing web applications with Flask.

Hello, Flask!

Download hello_flask.py or paste the following into a file named `hello_flask.py`:

```
1  from flask import Flask, request
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return "<h1>Hello, Flask!</h1>"
8
9  if __name__ == '__main__':
10     app.run(debug=True)
```

In the same directory as your `hello_flask.py` file run:

```
1  $ python3 hello_flask.py
2   * Running on http://127.0.0.1:5000/
3   * Restarting with reloader
```

If you see that output, you should be able to visit your web application in your browser at http://localhost:5000/

Initialization

All Flask applications must create an application instance:

```
1 from flask import Flask
2
3 app = Flask(__name__)
```

The argument to the Flask constructor is the name of the main module or package of the application. For our web apps it will always be `__name__`.

Routes and View Functions

Routes map URLs that a web site visitor sees in their address bar to a servier side resource. In:

```
1 @app.route("/")
2 def index():
3     return "<h1>Hello, Flask!</h1>"
```

- `@app.route("/")` registers the function below it, in this case index(), as the handler for / (the index, or default page)
- `@app.route()` is an example of a decorator function, which is a special syntax for higher-order functions (functions that take functions as parameters). Don't worry about the details.
- `index()` is an example of a view function.
- The string returned from a view function is sent in the reponse to the client

Dynamic Routes

Add this function to `hello_flask.py`

```
1 @app.route("/user/<name>")
2 def user(name):
3     return f"<h1>Hello, {name}!</h1>"
```

- `/user/` is the static part of the route. It must always appear for this view function to be called.
- `<name>` is the dynamic part of the route. It may change on each request, or even be absent
- `<name>` matches any text that appears after the static part of the route up to the next forward slash

Stop your hello_flask.py application with CTRL-C and restart it (if necessary), and visit http://localhost:5000/user/Lionel

Jinja2 Templates

In the previous examples our view functions returned strings that we generated directly in the functions. It's cleaner to use a template engine.

- A template is a text file that has placeholders for data to be inserted
- *Rendering* is the process of replacing the placeholders in a template with values
- Flask uses the Jinja2 template engine
- By default, Flask looks for templates in a subdirectory named templates

Download hello_jinja2.py and the templates directory.

Template Variables

Here's a simple template (templates/user.html.jinja2):

```
<html>
<head>
  <title>Hello, {{name}}</title>
<body>
  <h1>Hello, {{name}}</h1>
</body>
</html>
```

And a view function that renders it:

```
@app.route('/user/<username>')
def user(username):
    return render_template('user.html.jinja2', name=username)
```

- Keyword arguments to render_template specify key-value pairs for substitution in the template
- In this example, every instance of the variable `{{name}}` in the template is replaced with the value of `username` from the view function

Control Structures in Templates

Jinja2 supports control structures such as if statements:

```
{% if user %}
  Hello, {{ user }}!
{% else %}
  Hello, Stranger!
{% endif %}
```

and for loops:

```
<ul>
  {% for comment in comments %}
    <li>{{ comment }}</li>
  {% endfor %}
</ul>
```

Complete Example: Gradebook

Download gradebook.zip and extract it.

- In `grades.py` the `gradebook()` view function parses a CSV file from the local file system and passes data to the `grades.html.jinja2` template

```
1  @app.route("/grades/<course>/<term>")
2
3  def gradebook(course, term):
4      file_name = course + term + ".csv"
5      rows = []
6      with open(file_name, "r") as fin:
7          reader = csv.reader(fin)
8          for record in reader:
9              rows.append(record)
10     return render_template("grades.html.jinja2",
11         course=course, term=term, rows=rows)
```

- `grades.html.jinja2` uses nested for loops to populate an HTML table.

Take a look at the `grades.html.jinja2` template. How would it look if we used a `csv.DictReader`?

Closing Thoughts

- Tons more to know about web applications
- You know enough to make simple, yet useful web applications
- You have a big head start for CS 4400