CS 2316 Data Manipulation for Engineers

Flask

Christopher Simpkins
chris.simpkins@gatech.edu

Web Application Frameworks

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

First, you need the pip utility. Go to https://pip.pypa.io/en/latest/installing.html and follow the instructions.

- Note: on Unix (e.g., Linux, Mac OS X) remember to use `python3`

Then do:

```
$ pip3 install flask
```

On Windows it's just `pip`, not `pip3`. If installation is successful, this should work:

```
$ python3
Python 3.4.0 (v3.4.0:04f714765c13, Mar 15 2014, 23:02:41)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import flask
```

Now we're ready to create a Flask web application.

Hello, Flask!

Download hello_flask.py, or paste this into a file named `hello_flask.py`

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def index():
    return '<h1>Hello, Flask!</h1>'

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

Save the file to disk and in the same directory as the file run:

```
$ python3 hello_flask.py
 * Running on http://127.0.0.1:5000/
 * 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:

```
from flask import Flask

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:

```
@app.route('/')
def index():
    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

```
@app.route('/user/<name>')
def user(name):
    return '<h1>Hello, %s!</h1>' % name
```

- `/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.

Variables

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

```
<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', 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

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>
```

A Complete Example: Gradebook

Download the files and subdirectories in gradebook.

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

```
@app.route('/grades/<course>/<term>')
def gradebook(course, term):
    file_name = course + term + '.csv'
    rows = []
    with open(file_name, 'r') as fin:
        reader = csv.reader(fin)
        for record in reader:
            rows.append(record)
    return render_template('grades.html',
        course=course, term=term, rows=rows)
```

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

Take a look at the grades.html 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