Flask

A MICRO FRAMEWORK

PHIL LEONOWENS – CSCI 4830

What is Flask?

- Flask is a micro framework
- Python
- Uses the Werkzeug Python Library
- Renders HTML using Jinja2

Flask
web development,
one drop at a time

Methodology of Flask

- Minimal
- Supports external libraries
- Doesn’t scale great
- Subclassing
- Free decision making necessitates smart development

Werkzeug

- Python library for web servers
- Very minimal code
- Human readable
- Primarily used as Backbone of Flask

Jinja2

- Templating Language
- Can embed python code within html
- Makes handling of requests easy
- Very simple to implement existing Python Code

Jinja2

- Templates are all inside the templates subfolder.
- Dictionary elements are called with dot syntax

```
<h4> User: </h4><p> {{ entry.username }}</p>

<h4> Tweet: </h4><p> {{ entry.post }}</p>
```

Is equal to entry[‘username’] and entry[‘post’]

- Conditionals can be called

```
{% if x > 5 %}

#Do stuff

{% endif %}
```

Jinja2

- Includes

```
{% include'/navbar.html' %}
```

- Function Calls

```
Score: <p> {{ entry.score|round(2)  }}</p>
```

Equal to

```
entry[‘score’].round(2)
```

- Simple Python Syntax

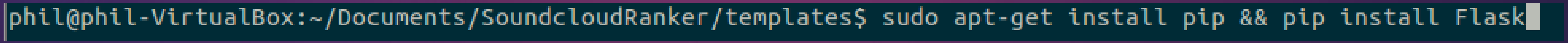
```
Score: <p> {{ ((entry.score*5)+5)|round(2)  }} /10</p>
```

Installing Flask

- Very Simple

```
sudo apt-get install pip
pip install Flask
```

- Flask apps run on port 5000



```
phil@phil-VirtualBox:~/Documents/SoundcloudRanker/templates$ sudo apt-get install pip && pip install Flask
```

Making a quick app

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

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

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

What does Flask make easy for us?

Handling Routes

- The route decorator is used to handle routes

```
@app.route('/')
def mainPage(entries=None):
    return render_template('main.html', entries= [‘John’,
‘Bob’])
@app.route('/songs/')
def songsPage(theEntries= getEntries()):
    entryManager.sortEntries()
    return render_template('songs.html', entries= theEntries)
```

Handling Routes

localhost:5000/songs/50

```
@app.route('/songs/<songid>')
def song(songid=None):
    if songid is not None:
        return render_template('song.html', entry=getEntry(songid))
    else:
        return render_template('songs.html', entries= getEntries())
```

url_for

- url_for can be used to create easy urls you don't have to remember
- Generates URLs based on function names

```
url_for('index')
```

- Can also pass in arguments

```
url_for('profile', username='John Doe')
```

- Can be used for static pages

```
stylePage = url_for('static', filename='style.css')
```

Benefits

- No need to remember URLs
- Escapes special characters
- Can handle urls outside of root dir

HTTP Requests (GET)

```
@app.route('/actions', methods=['GET'])
def actions():
    if request.method == 'GET':
        if (request.args.get('action') == 'update'):
            entryManager.update()
        if (request.args.get('action') == 'delete'):
           entryManager.removeFiles()
   else:
            error = 'No actions'
    return redirect(request.referrer)
```

HTTP Requests (POST)

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        login()
    else:
        show_the_login_form()
```

HTTP Requests (Other)

- HEAD

HEAD retrieves only the header information without passing you the whole request

- PUT

Not as widely used as POST but still an option and in some cases more useful due to multiple procedure calls

- DELETE

Remove the information at the given location, typically POST is used.

- OPTIONS

Allows the client to figure out which methods you support for this URL. This is implemented automatically by Flask.

Templates

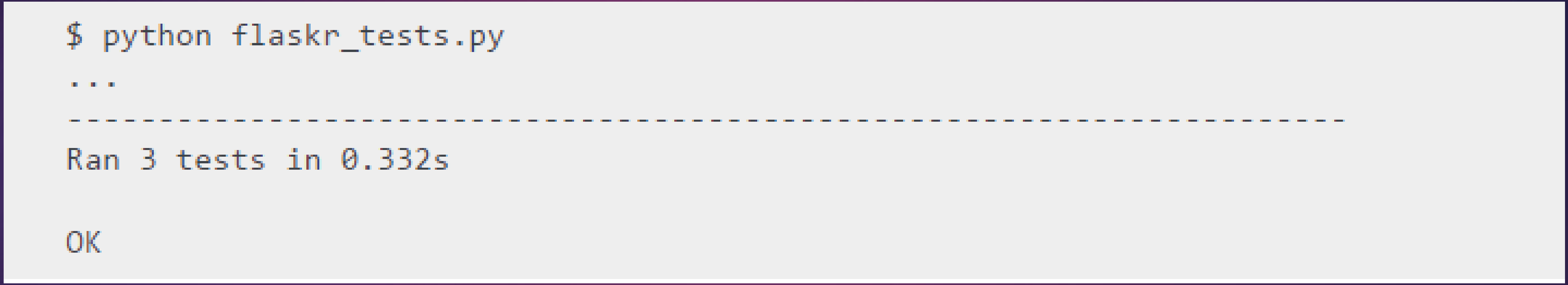
```
@app.route('/')
def mainPage(entries=None):
    return render_template('main.html', entries= getEntries())

@app.route('/songs/')
…
    return render_template('songs.html', entries= getEntries())
```

Testing

- Testing is made easy in Flask by using Flaskr
- We can use the requests module to accomplish this
- Similar to many other testing frameworks
  - def setUp(self) says what should be done at the beginning of the test
  - def tearDown(self) says what should be done at the end of the test
- Asserts are used to ensure that the proper behavior is being observed

Testing

```
class FlaskrTestCase(unittest.TestCase):
    def test_get_request(self)
        r = requests.get('http://localhost:5000/contacts/')
            assert 'contacts' in r.text
    def test_post_request(self):
        payload = 
{"name":"Putin","email":"putin@gmail.com","phone":"77777777"}
        r = requests.post('http://localhost:5000/contacts/', data=payload)
        assert r.status_code == requests.codes.ok
```



```
$ python flaskr_tests.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.332s

OK
```

Other Useful Features

- Debugging Mode
  - Enables on the fly modification of code
  - Automatically Applied
  - Interactive Console
- Requests
  - request.args.get(‘username', '')
- Error Handler
- @app.errorhandler(404)
- def not_found(error):
  - return render_template('error.html'), 404

Deployment

- Amazon Web Services
- Heroku
- Google App engine
- Any Python hosting framework

References

- http://flask.pocoo.org/docs/0.10/
- jinja.pocoo.org/docs/dev/
- werkzeug.pocoo.org/