Coding Bootcamp Code in Python

EXPOSING YOUR MODEL: FLASK

Introduction

- Micro-services
  - single user web application
  - can act as API (REST interface)
  - can act as GUI (HTML/Javascript)
- Many frameworks, e.g., Flask
  - easy to install
  - nice for simple applications
- Note: security!

Did I mention that you should really, *really* know what you are doing?

Do not (and I mean *never ever*) use this for production or die!

Hello Flask

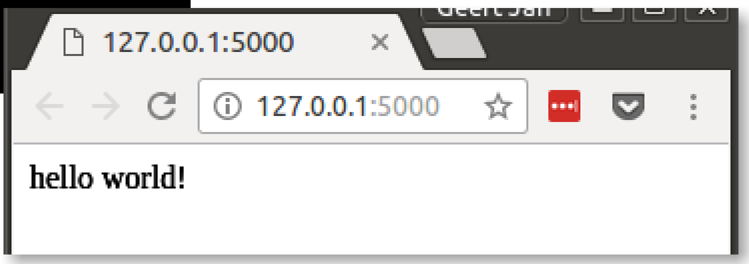
- Web application

```
from flask import Flask

app = Flask(__name__)

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

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

Handles request for /

- Run

```
$ python ./hello_world.py
 * Running on http://127.0.0.1:5000/
   (Press CTRL+C to quit)
```



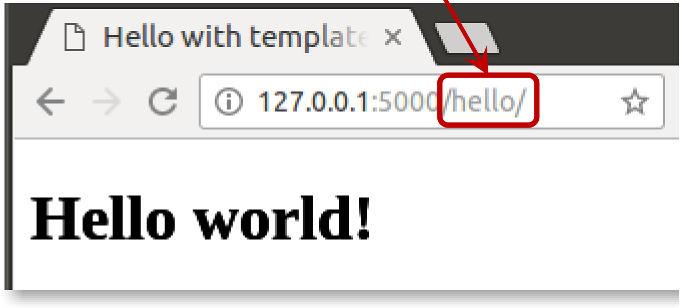
hello world!

Hello yourself

- Routing with variables

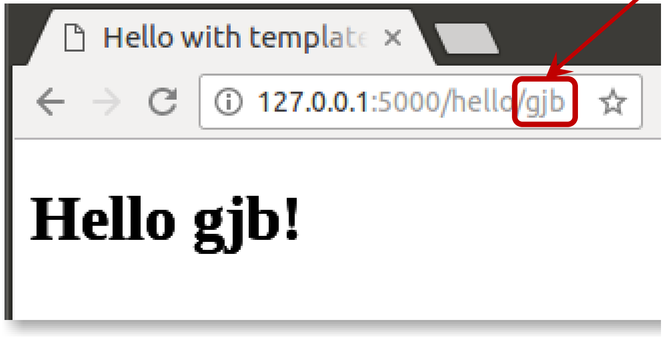
```
@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return 'hello {0}!'.format(name if name else 'world')
```

default route: `name == None`



Hello world!

variable



Hello gjb!

Variable types: `int`, `float`, `path`, `uuid`

Templates

- Templates: Flask uses jinja2

templates/hello.html

```
<!doctype html>

<title>Hello with templates</title>

<body>
{% if name %}
    <h1>Hello {{name}}!</h1>
{% else %}
    <h1>Hello world!</h1>
{% endif %}
<p> This is a jinja2 template. </p>
</body>
```

variable substitution

if name defined

if name undefined

- Using template

```
from Flask import render_template
…
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Form template

- Form template (ugly, use CSS for style)

templates/sum.html

```
<!doctype html>
<title>EasySum</title>
<body>
    <h1>EasySum</h2>
    {% if result %}
        <p> The result of {{op1}} + {{op2}} is {{result}}. </p>
    {% else %}
        <p> This web aplication computes the sum of two numbers
          , the can even be floating point values. </p>
        <form method="POST">
            First operand:  <input type="text" name="op1"/> <br/>
            Second operand: <input type="text" name="op2"/> <br/>
            <input type="submit" value="Compute"/>
        </form>
    {% endif %}
</body>
```

Form handling

- Use `request` & specify `methods`

```
from flask import Flask, request, render_template
…
@app.route('/', methods=['GET', 'POST'])
def start():
    if request.method == 'POST':
        op1 = float(request.form['op1'])
        op2 = float(request.form['op2'])
        result = op1 + op2
        return render_template('sum.html', op1=op1, op2=op2,
                                           result=result)
    else:
        return render_template('sum.html')
```

Methods to handle

Check method

Retrieve input values

- Note: *always* validate input!

Sessions

- Keep track of application state with `session`

```
app.secret_key = 'A1Zr38g/3yZ R~XGH!jlN]3WX/,?RT'

@app.route('/reset')
def reset():
    session['fib'] = 1
    session['fib_prev'] = 1
    session['number'] = 0
    return render_template('fib.html', number=None)

@app.route('/')
def start():
    if 'number' in session:
        session['number'] += 1
        fib = session['fib']
        session['fib'] += session['fib_prev']
        session['fib_prev'] = fib
        return render_template('fib.html',
                               number=session['number'],
                               fibonacci=session['fib'])
    else:
        return reset()
```

Persistent variables

Note: Flask server is single session
⇒ no multi-user, multi-session!

Check state

Retrieve/update session values

Further reading

- Flask
  http://flask.pocoo.org/docs/latest
- Jinja2

  http://jinja.pocoo.org/docs/latest
- Did I mention security is an issue?
  http://flask.pocoo.org/docs/latest/advanced_foreword/#develop-for-the-web-with-caution

There be dragons!

pickle module

Python pickle module is used for serializing and de-serializing a Python object structure.

- Any object in Python can be pickled so that it can be saved on disk.
- The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

```python
import pickle

def storeData():
    # initializing data to be stored in db
    Ricardo = {'key' : 'Ricardo', 'name' : 'Ricardo Profile'
,
    'age' : 43, 'pay' : 40000}
    Pinto = {'key' : 'Pinto', 'name' : 'Pinto Profile',
    'age' : 50, 'pay' : 50000}

    # database
    db = {}
    db['Ricardo'] = Ricardo
    db['Pinto'] = Pinto

    # Its important to use binary mode
    dbfile = open('examplePickle', 'ab')

    # source, destination
    pickle.dump(db, dbfile)
    dbfile.close()

def loadData():
    # for reading also binary mode is important
    dbfile = open('examplePickle', 'rb')
    db = pickle.load(dbfile)
    for keys in db:
        print(keys, '=>', db[keys])
    dbfile.close()

if __name__ == '__main__':
    storeData()
    loadData()
```

Code Pack 31

Full App