Web Development with Python

AY250 Fall 2013

```
git pull; pip install flask, cherrypy
```

http://www.linuxforu.com/how-to/django-when-python-bites-the-web/

authors: C. Stark, C. Klein, J. Bloom

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

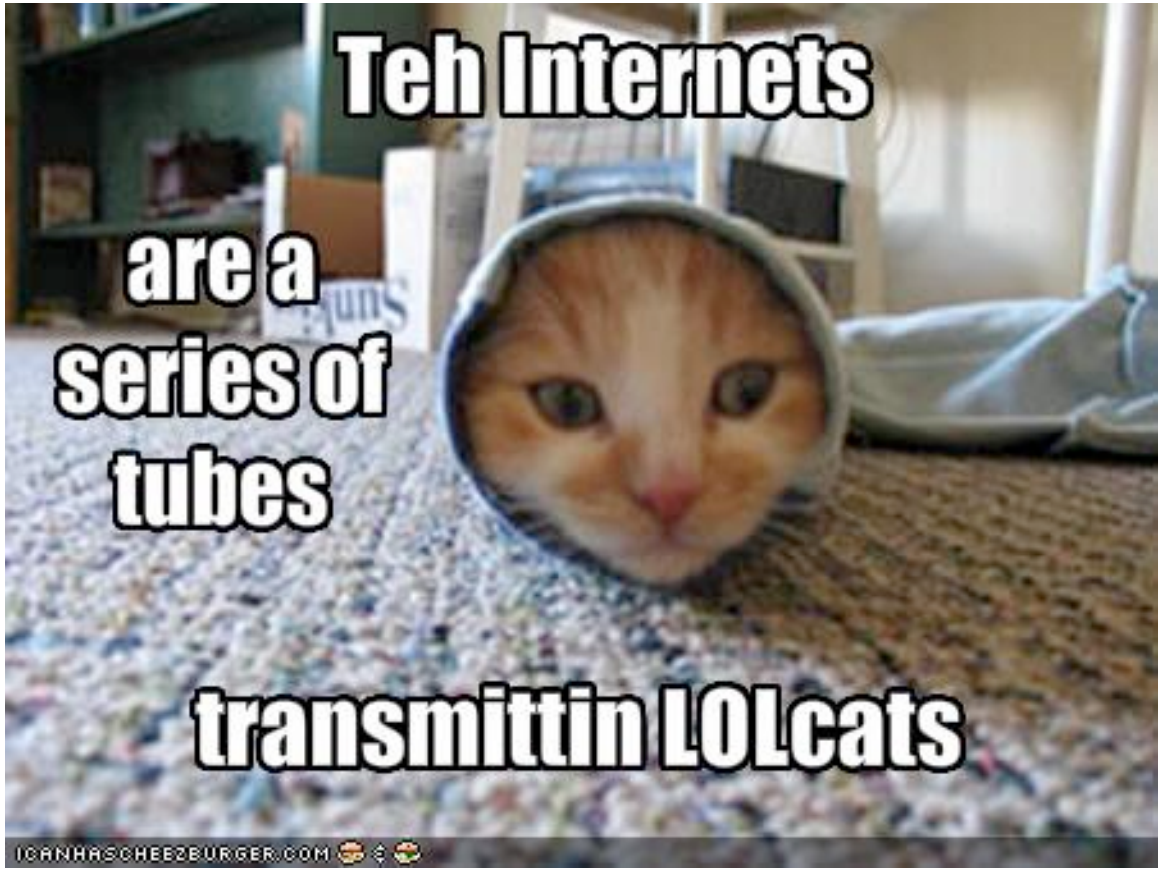
Overview of Today's Lecture

The web paradigm

Using Python for web supremacy

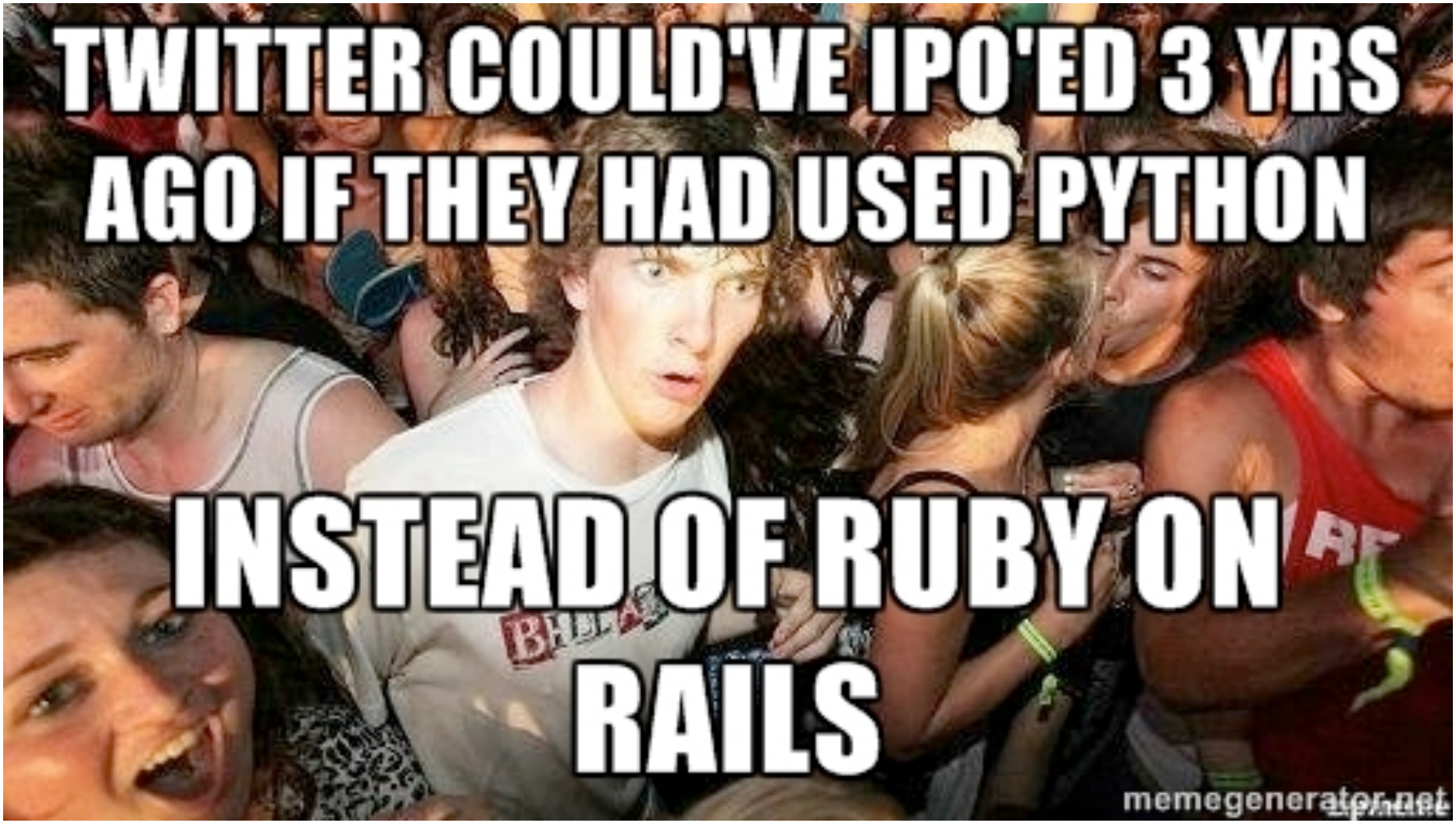
- Basic Python servers
- Frameworks and using Flask
- Google App Engine

Believe it or not



the internet was not built for this.

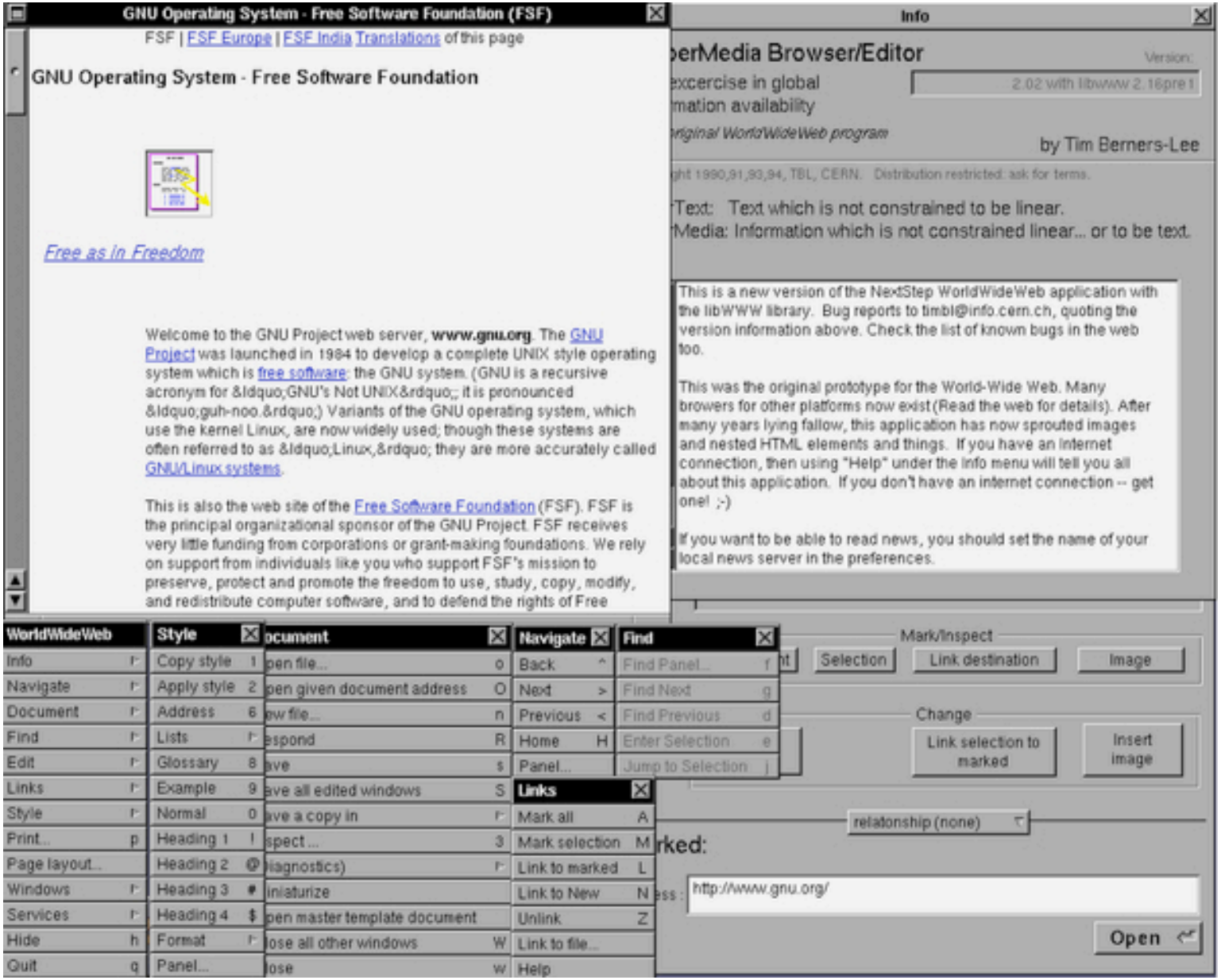

I SHOULD BUILD MY ENTIRE WEBSTACK
WITH PYTHON
megenerator.net



http://i.imgur.com/hAvEgsA.jpg

It **was** actually built for science (and defense)

The browser model



http://en.wikipedia.org/wiki/Web_browser

Request and Response

User knocks on the server's door

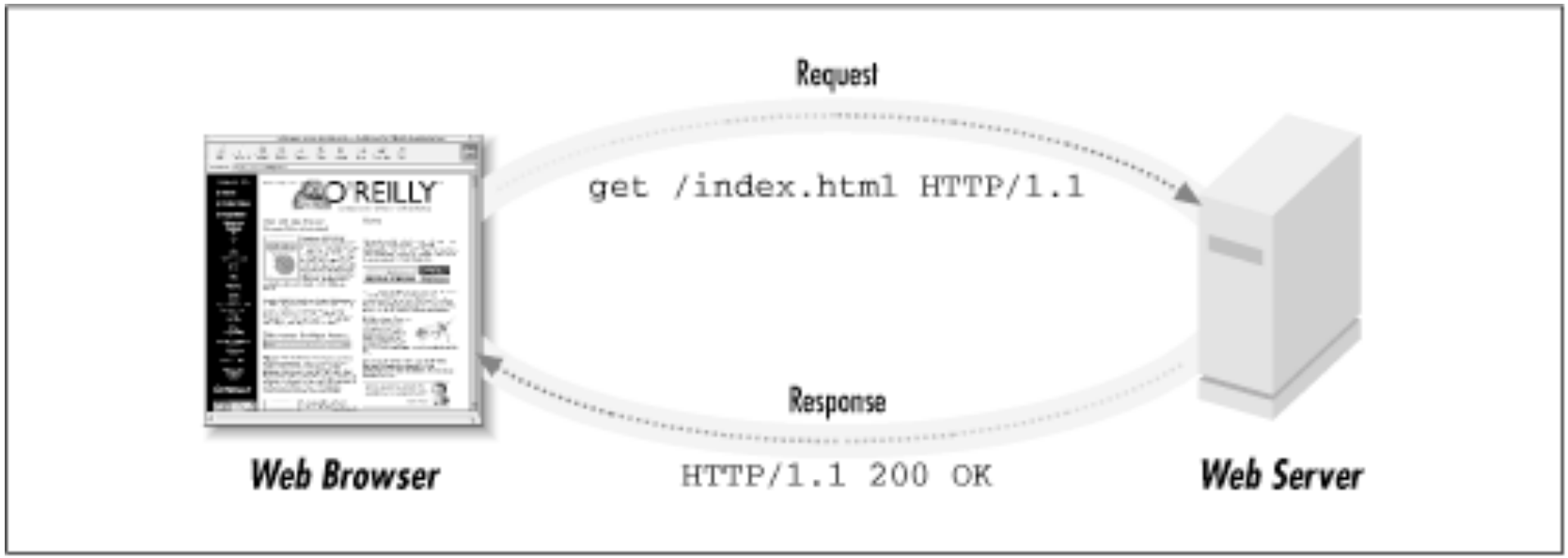
(request)

Server thinks about it

Server sends data back to the user

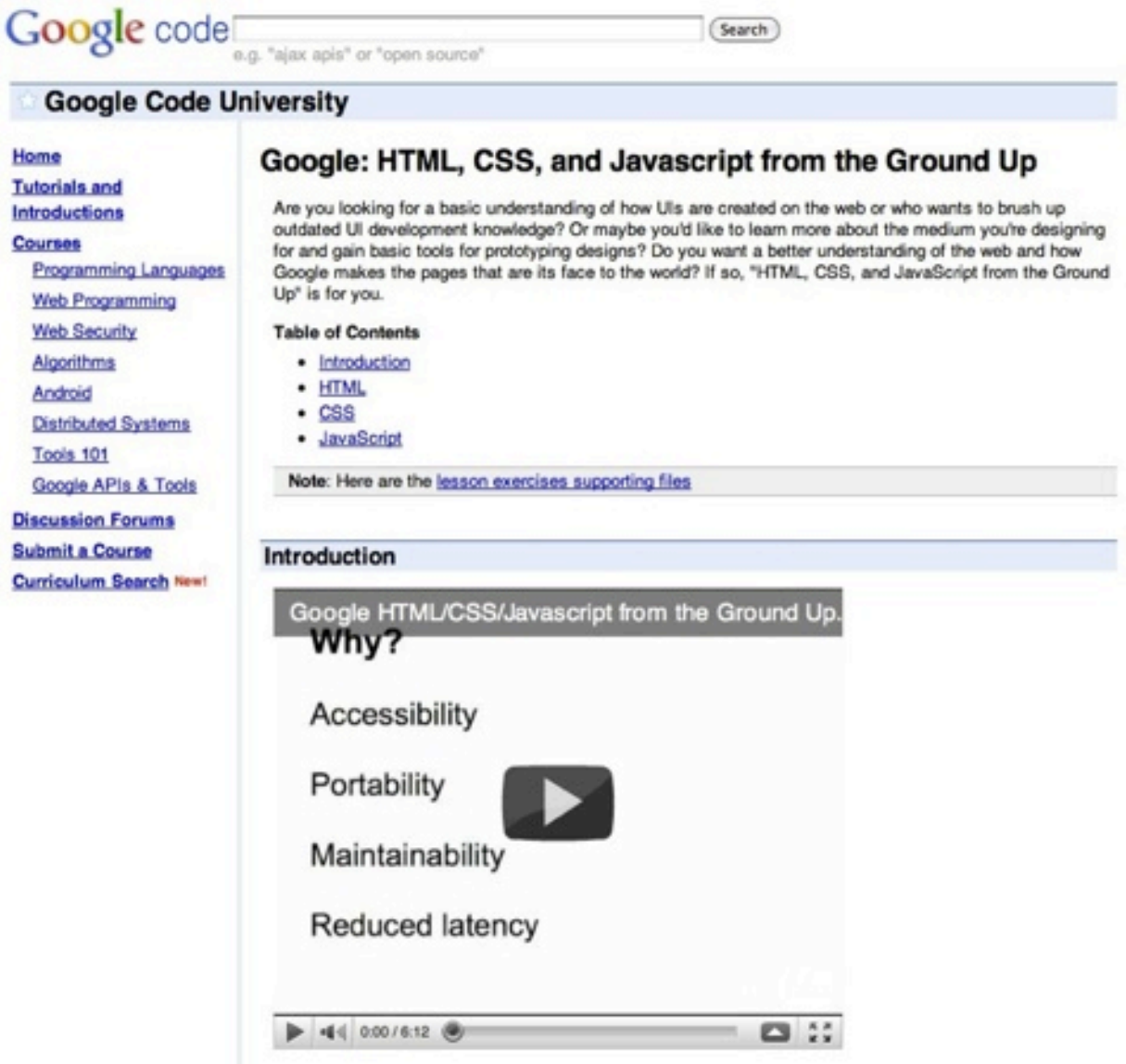
(response)

Request and Response



http://stein.cshl.org/~lstein/talks/perl_conference/apache_api/lifecycle.html

Browser stuff



http://code.google.com/edu/ajax/tutorials/intro-to-js.html

Browser stuff

Hyper Text Markup Language (HTML)	Structure and content
Cascading Style Sheet (CSS)	Presentation
JavaScript	Behavior (dynamic stuff)

Browser stuff

Python helps generate the (html) content.

In general, you work on CSS and JS separately.

Finally, use Python to serve all of this media to the user.

Simple Servers

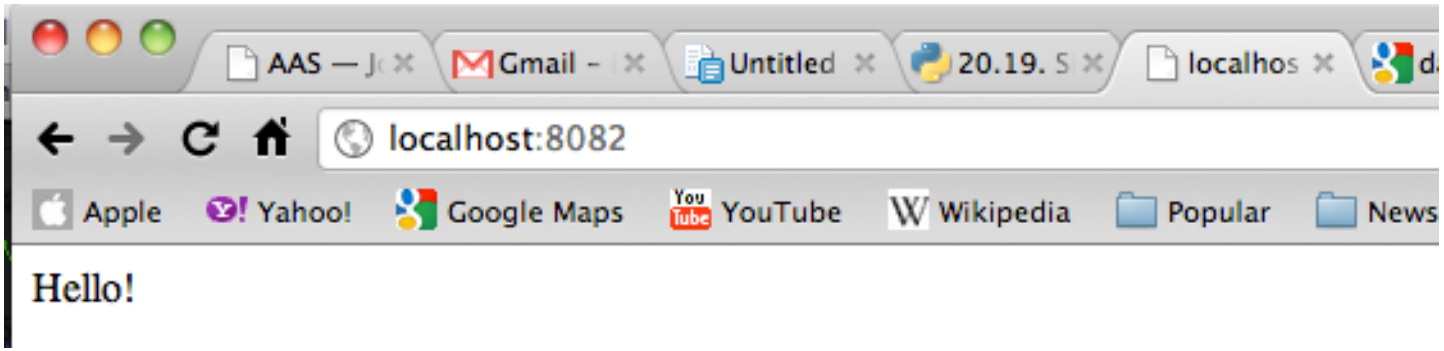
Recall the XML/RPC Server...

a webserver just responds to a different request protocol

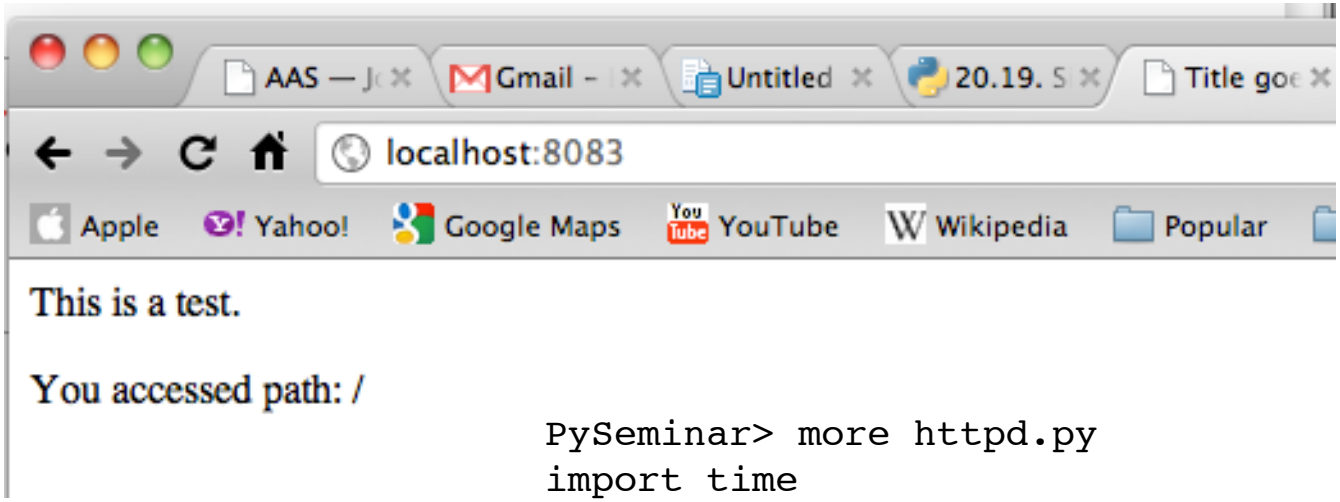
```
import BaseHTTPServer
class myresponse(BaseHTTPServer.BaseHTTPRequestHandler):
   def do_GET(s):
       s.wfile.write("<body>Hello!</body>")

httpd = BaseHTTPServer.HTTPServer(("localhost", 8082), myresponse)
httpd.serve_forever()
```

`wfile`: output stream file

note: BaseHTTPServer is not threaded

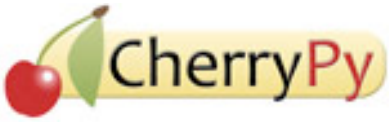


```
Py4Science> python httpd.py
Sat Nov  3 21:29:43 2012 Server Starts - localhost:8083
1.0.0.127.in-addr.arpa - - [03/Nov/2012 21:30:00] "GET / HTTP/1.1" 200 -
1.0.0.127.in-addr.arpa - - [03/Nov/2012 21:30:00] "GET /favicon.ico HTTP/1.1" 200 -
^CSat Nov  3 21:32:59 2012 Server Stops - localhost:8083
```




```
PySeminar> more httpd.py
import time
import BaseHTTPServer
HOST_NAME = 'localhost' # !!!REMEMBER TO CHANGE THIS!!!
PORT_NUMBER = 8083 # Maybe set this to 9000.
class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_HEAD(s):
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()
    def do_GET(s):
        """Respond to a GET request."""
         ....
```

Simple Servers

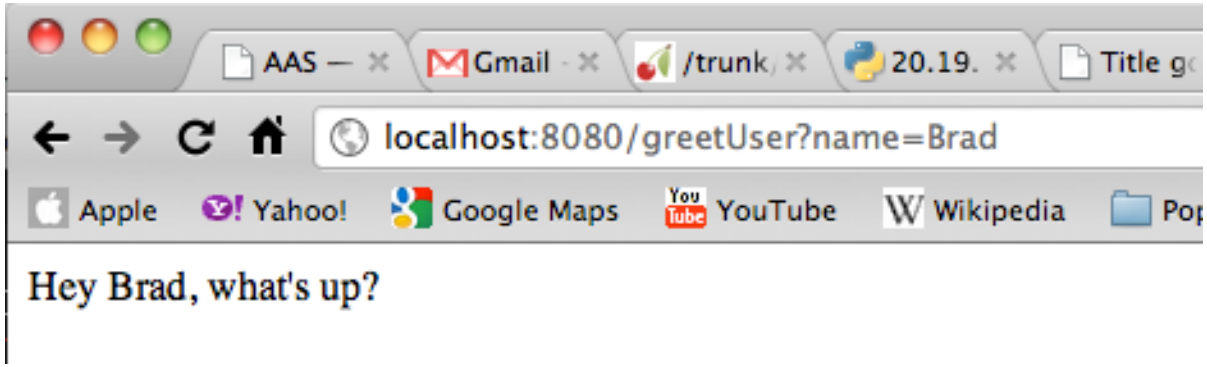


: `pip install cherrypy`

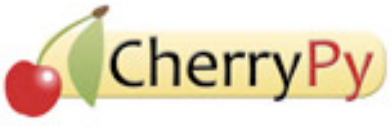
file: cp1.py

```
import cherrypy
class WelcomePage:
    def greetUser(self, name = None):
        if name:
            # Greet the user!
            return "Hey %s, what's up?" % name
        else:
            return 'call me like <i>http://localhost:8080/greetUser?name=Josh</i>'
    greetUser.exposed = True

cherrypy.quickstart(WelcomePage())
```

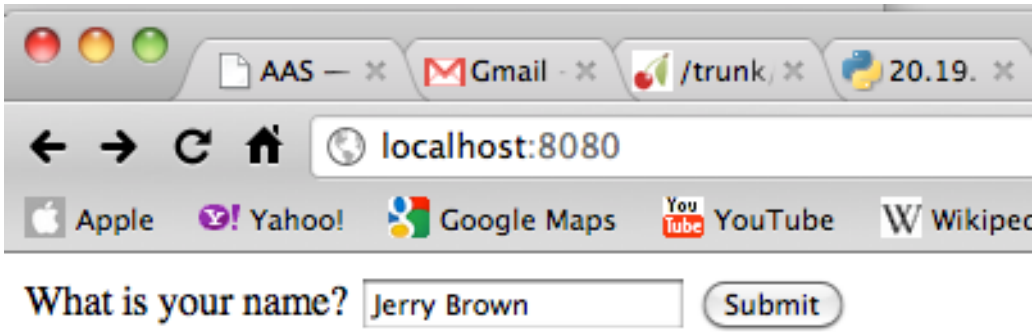
here, `name` is a variable of the GET request



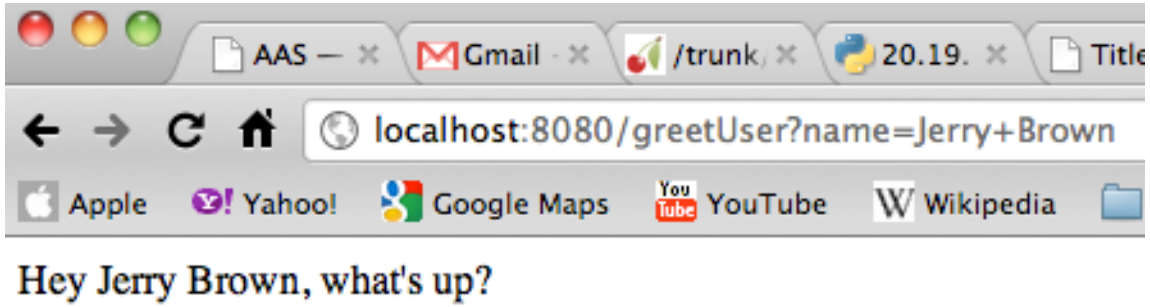
Simple Servers



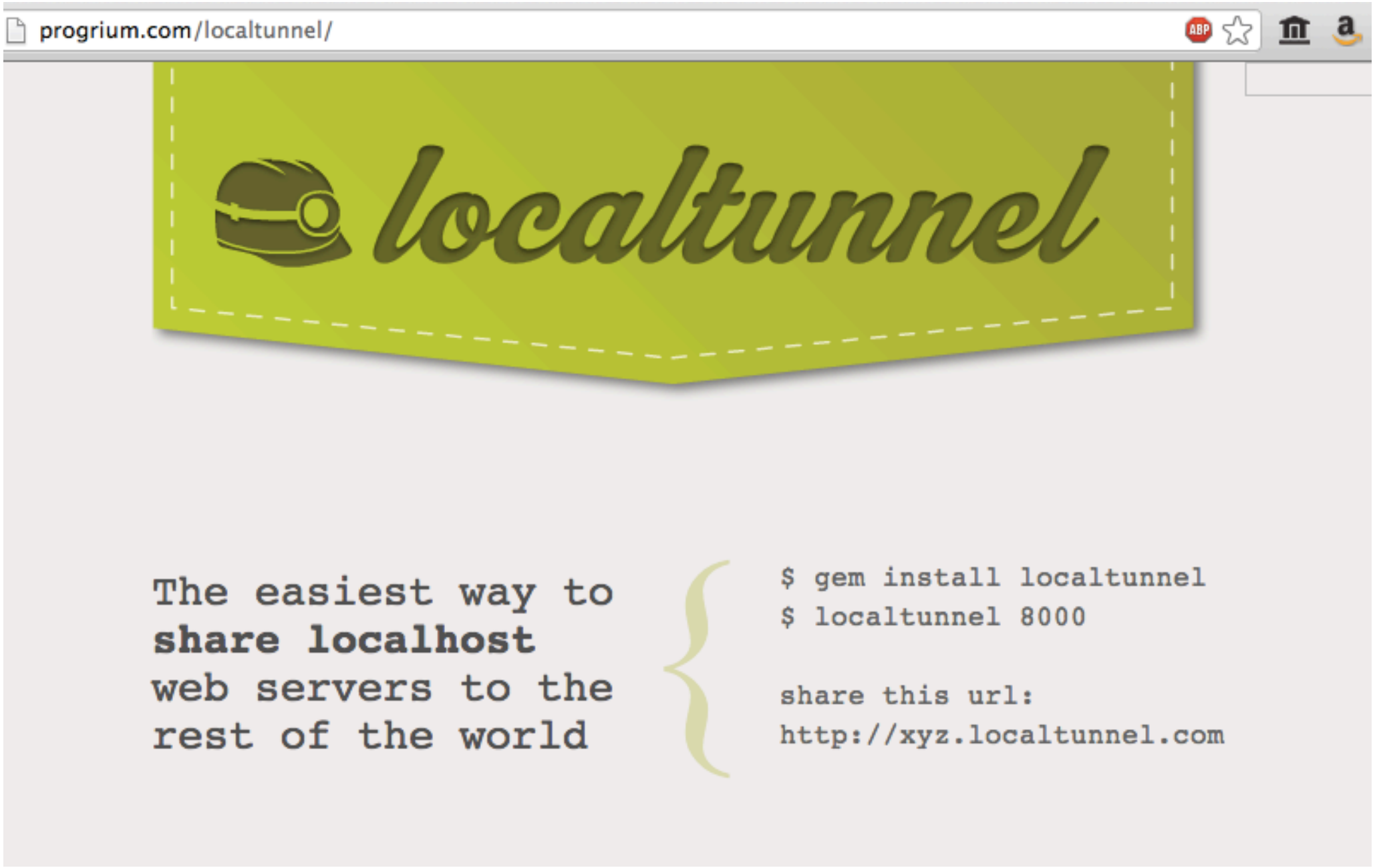

```
>>> run cp2
```




```
def index(self):
    # Ask for the user's name.
    return '''
        <form action="greetUser" method="GET">
        What is your name?
        <input type="text" name="name" />
        <input type="submit" />
        </form>'''
index.exposed = True
```



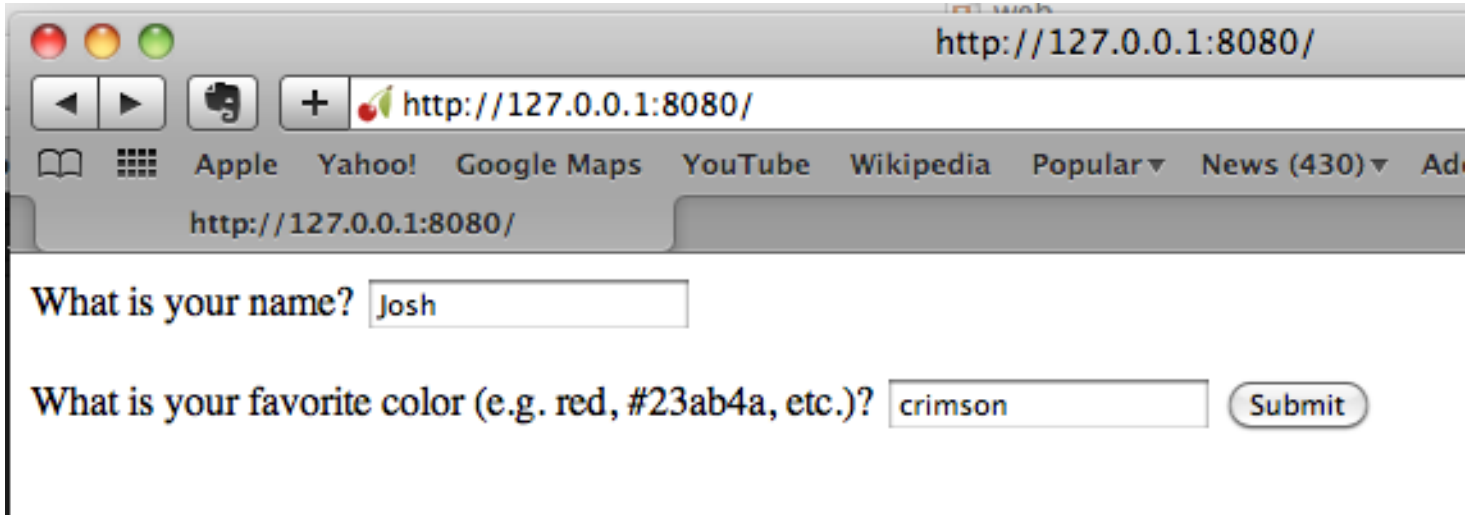
localtunnel -k ~/.ssh/id_rsa.pub 8083

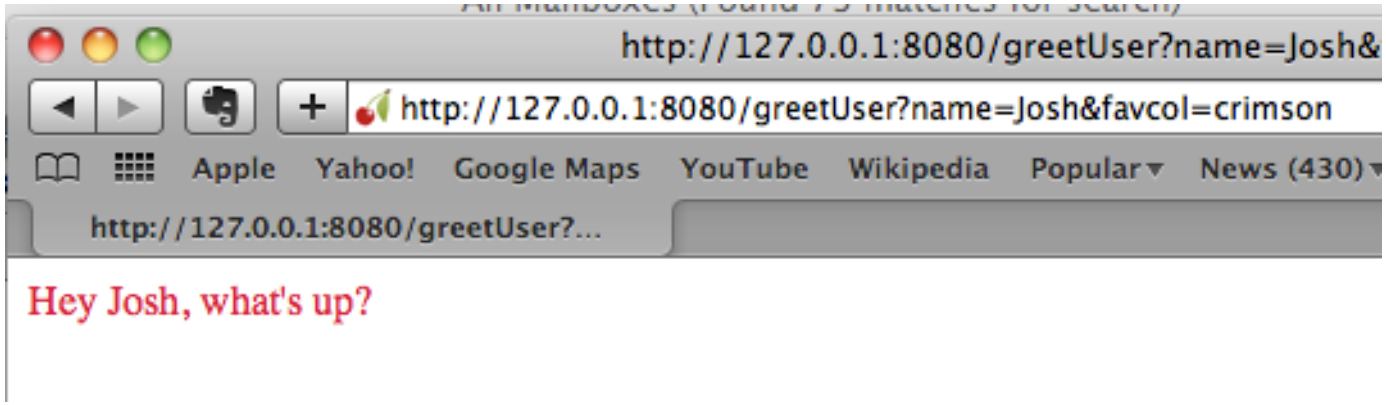


Small exercise:

modify `cp2.py` so that it asks the user for their name and their favorite color. Then greet them with that color...

```
<font color="red">output</font>
```





What's WSGI?

HTTP = HyperText Transfer Protocol

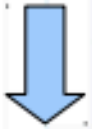
WSGI = Web Server Gateway Interface

http://www.python.org/dev/peps/pep-0333/

http://gustavonarea.net/files/talks/europython2010/wsgi-from-start-to-finish.pdf

HTTP and WSGI requests

```
POST /login HTTP/1.1
Host: example.org
User-Agent: EP2010 Client
Content-Length: 25
          empty line
username=foo&password=bar
```



```
{
'REQUEST_METHOD': “POST”,
'PATH_INFO': “/login”,
'SERVER_PROTOCOL: “HTTP/1.1”,
'HTTP_HOST': “example.org”,
'HTTP_USER_AGENT': “EP2010 Client”,
'CONTENT_LENGTH': “25”,
'wsgi.input': StringIO(“username=foo&password=bar”),
}
```

http://gustavonarea.net/files/talks/europython2010/wsgi-from-start-to-finish.pdf

HTTP and WSGI responses

```
HTTP/1.1 200 OK
Server: EP2010 Server
Content-Length: 18
Content-Type: text/plain
          empty line
Welcome back, foo!
```

```
(
  “200 OK”,
  [
    (“Server”, “EP2010 Server”),
    (“Content-Length”, “18”),
    (“Content-Type”, “text/plain”),
  ]
)

[“Welcome back, foo!”]
```

Note that:

- It's not a single object.
- The HTTP version is not set.

Use any **WSGI compliant server** to serve up your app.
A server may have been written to support different kinds of features:

 * speed, performance (multi-threaded, written in a compiled language like C)
 * extensibility (written in pure Python)
 * ease of development (code reloading, extra debugging features)
 * adapters for a larger server platform (e.g. mod_wsgi for Apache, or the WSGI adapter for Google App Engine)

Depending on what you need for development or deployment you can pick a server that matches your needs best.

https://bitbucket.org/lost_theory/wsgitalk

http://lucumr.pocoo.org/2011/7/27/the-pluggable-pipedream/

modwsgi

Python WSGI adapter module for Apache.

Project Information

+1 +43 Recommend this on Google

Starred by 683 users

Project feeds

Code license

Apache License 2.0

Labels

Python, Apache, WSGI

Members

Graham.Dumpleton@gmail.com, mark.joh...@gmail.com

Featured

Downloads

mod_wsgi-3.4.tar.gz

Show all »

Wiki pages

ChangesInVersion0304

Show all »

Links

What Is mod_wsgi? ¶

The aim of mod_wsgi is to implement a simple to use Apache module which can host any Python application which supports the Python WSGI interface. The module would be suitable for use in hosting high performance production web sites, as well as your average self managed personal sites running on web hosting services.

Modes Of Operation

When hosting WSGI applications using mod_wsgi, one of two primary modes of operation can be used. In 'embedded' mode, mod_wsgi works in a similar way to mod_python in that the Python application code will be executed within the context of the normal Apache child processes. WSGI applications when run in this mode will therefore share the same processes as other Apache hosted applications using Apache modules for PHP and Perl.

An alternate mode of operation available with Apache 2.X on UNIX is 'daemon' mode. This mode operates in similar ways to FASTCGI/SCGI solutions, whereby distinct processes can be dedicated to run a WSGI application. Unlike FASTCGI/SCGI solutions however, neither a separate process supervisor or WSGI adapter is needed when implementing the WSGI application and everything is handled automatically by mod_wsgi.

Because the WSGI applications in daemon mode are being run in their own processes, the impact on the normal Apache child processes used to serve up static files and host applications using Apache modules for PHP, Perl or some other language is much reduced. Daemon processes may if required also be run as a distinct user ensuring that WSGI applications cannot interfere with each other or access information they shouldn't be able to.

Note that although mod_wsgi has features similar to FASTCGI/SCGI solutions, it isn't intended to be a replacement for those hosting mechanisms in all situations for Python web hosting. Specifically, mod_wsgi is not designed for nor intended for use in over allocated shared mass virtual hosting setups for different users on a single Apache instance. For such mass virtual hosting arrangements, FASTCGI in particular would still be the preferred choice in most situations.

```
29  import pprint
30
31  def application(environ, start_response):
32      start_response('200 OK', [('content-type', 'text/html')])
33      yield "<pre>\n"
34      yield pprint.pformat(environ)
35      yield "\n</pre>"
36
37  if __name__ == '__main__':
38      import sys
39      arg = sys.argv.pop(-1)
40
41      if arg == 'wsgiref':
42          from wsgiref.simple_server import make_server
43          print "Serving on http://localhost:4000..."
44          make_server('localhost', 4000, application).serve_forever()
45
46      elif arg == 'werkzeug':
47          from werkzeug import run_simple
48          run_simple('localhost', 4000, application, use_debugger=True)
49
50      elif arg == 'cherrypy':
51          from cherrypy import wsgiserver
52          server = wsgiserver.CherryPyWSGIServer(('localhost', 4000), application)
53          print "Serving on http://localhost:4000..."
54          try:
55              server.start()
56          except KeyboardInterrupt:
57              print 'Shutting down.'
58              import sys; sys.exit();
59
60      else:
61          print '''Please provide one of:
62  * wsgiref
63  * werkzeug
64  * cherrypy'''
```

Frameworks

Don’t spend time writing code for common tasks.

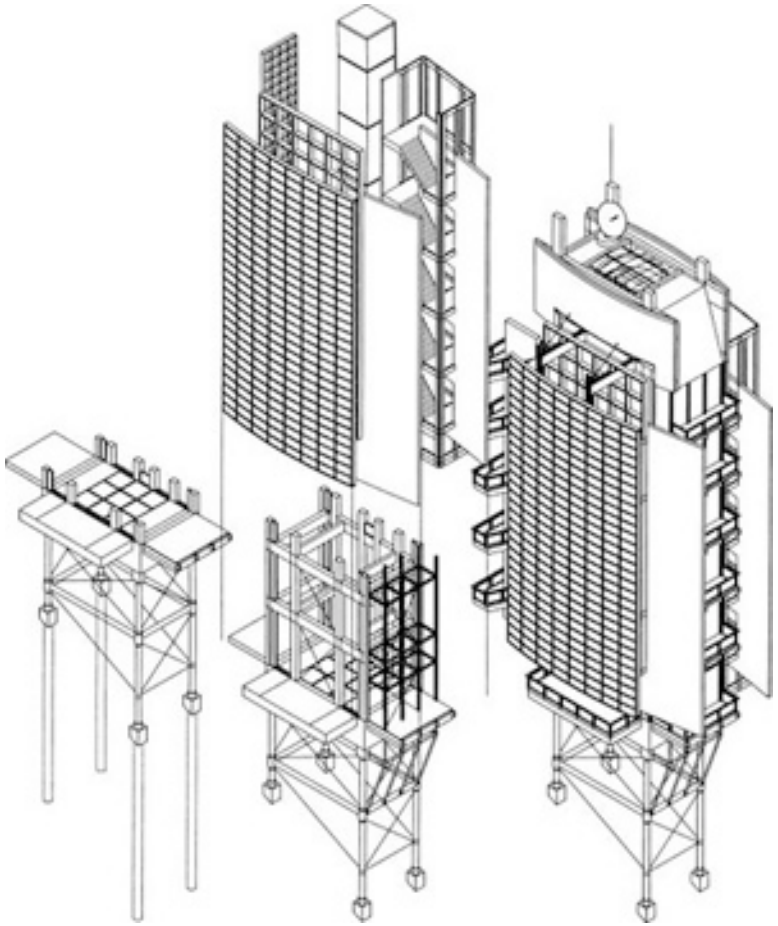
Query database B for activity of user 3...

Check that submitted form field is int...

Design another 1 - 5 star rating system...

Teams of people much more experienced than us have worked on this stuff for years.

Frameworks



Assume some architecture and build with the pieces you are given.

http://www.thebaa.org.uk/

The (Python) framework world

Web micro-framework battle

http://www.youtube.com/watch?v=AYjPIMe0BhA

The (Python) framework world

Flask

web development,
one drop at a time

http://flask.pocoo.org/

pip install flask

Hello World

fhello.py

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

@app.route("/")
def hello():
    return "Hello World!"

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

```
Py4Science> python fhello.py
 * Running on http://127.0.0.1:5000/
```

URL Route Registration

Where users can go to get things on your site
using "view decorators" of "view functions"

```
@app.route('/')
....
@app.route('/hello')
...
@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % username
```

```
@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id
```

urls.py

HTTP Methods

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

methods.py

Forms

form1.py

```
@app.route('/welcome', methods=['GET', 'POST'])
def welcome():

    if request.method == 'POST':
        username = request.form['name']
        if username not in ("","  ",None):
            return "Hey %s, what's up?" % username
        else:
            return """We really want to know your name. Add it
                      <a href='%s'>here</a>""" % url_for("welcome")
    else:
        ## this is a normal GET request
        return '''
            <form action="welcome" method="POST">
            What is your name?
            <input type="text" name="name" />
            <input type="submit" />
            </form>'''
```

but it's annoying to have to put HTML into Python...

Templates

What users see. You need to know HTML to make these.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Super Site</title>
    </head>

    <body>
        <h1>{{ page_title }}</h1>

        <p>This content is “dynamic”:</p>
            {% block content %}{% endblock
            %}
    </body>
</html>
```

templates/base.html

```
1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4  @app.route("/")
5  def hello():
6      return render_template('base.html',
7                 page_title="Templates",
8                 content="hello!")
9
10 if __name__ == "__main__":
11     app.run()
```

thello.py

Templates

What users see. You need to know HTML to make these.

```
{% extends "base.html" %}

{% set page_title = 'My Form' %}

{% block content %}
            <form action="welcome"
method="POST">
            What is your name?
            <input type="text"
name="name" />
            <input type="submit" />
            </form>
{% endblock %}
```

templates/form.html

```
@app.route('/welcome', methods=['GET', 'POST'])
def welcome():

    if request.method == 'POST':
        username = request.form['name']
        if username not in ("","  ",None):
            return "Hey %s, what's up?" % user
        else:
            return """We really want to know y
                    <a href='%s'>here</a>"""
    else:
        ## this is a normal GET request
        return render_template("form.html")

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

thello.py

flask uses [Jinja2 templating]

Table Of Contents

Template Designer Documentation

This document describes the syntax and semantics of the template engine and will be most useful as reference to those creating Jinja templates. As the template engine is very flexible the configuration from the application might be slightly different from here in terms of delimiters and behavior of undefined values.

Synopsis

A template is simply a text file. It can generate any text-based format (HTML, XML, CSV, LaTeX, etc.). It doesn't have a specific extension, `.html` or `.xml` are just fine.

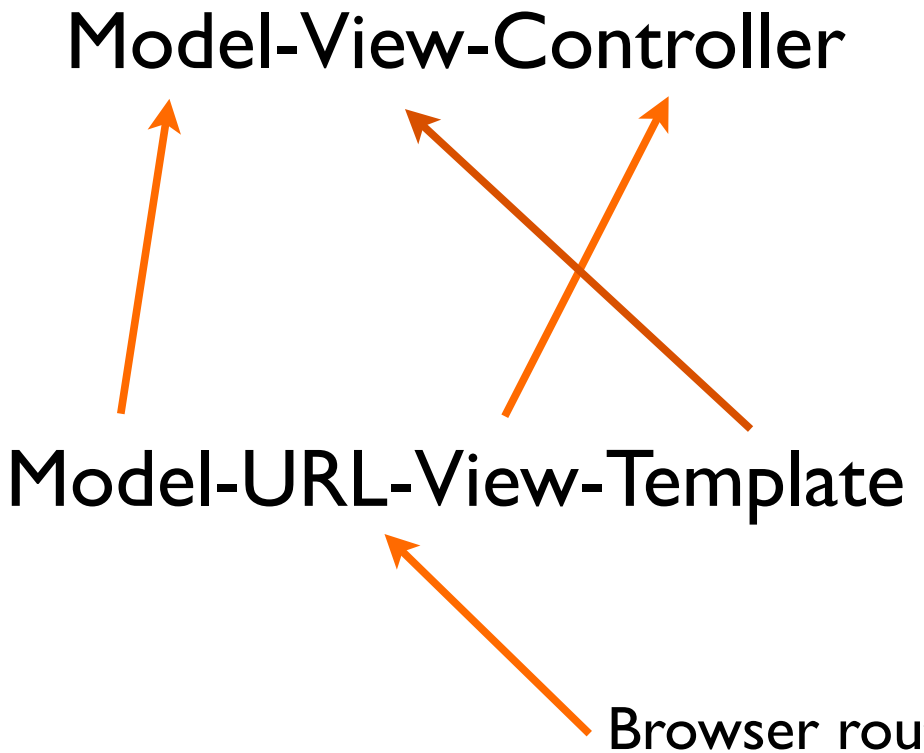
A template contains **variables** or **expressions**, which get replaced with values when the template is evaluated, and tags, which control the logic of the template. The template syntax is heavily inspired by Django and Python.

Below is a minimal template that illustrates a few basics. We will cover the details later in that document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
    {% for item in navigation %}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
    {% endfor %}
    </ul>

    <h1>My Webpage</h1>
```

Flask's MVC-like...

Remember?



Flask can behave something like it
with SQLAlchemy...

```
sudo pip install Flask-SQLAlchemy
```

Models: This is where data goes.

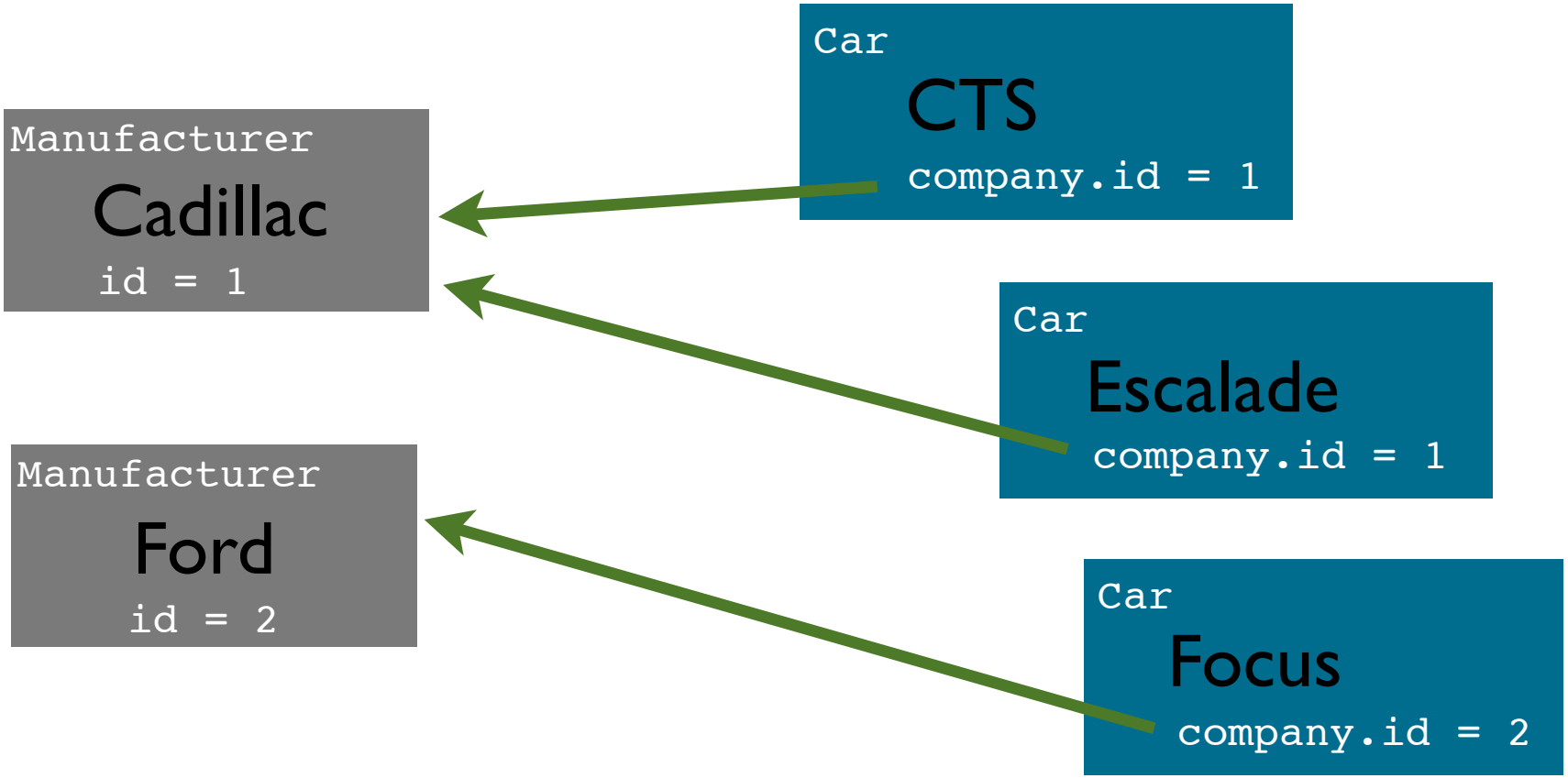
The classes that "model" the data objects that make up your app.

Stored in whatever database your config sets.

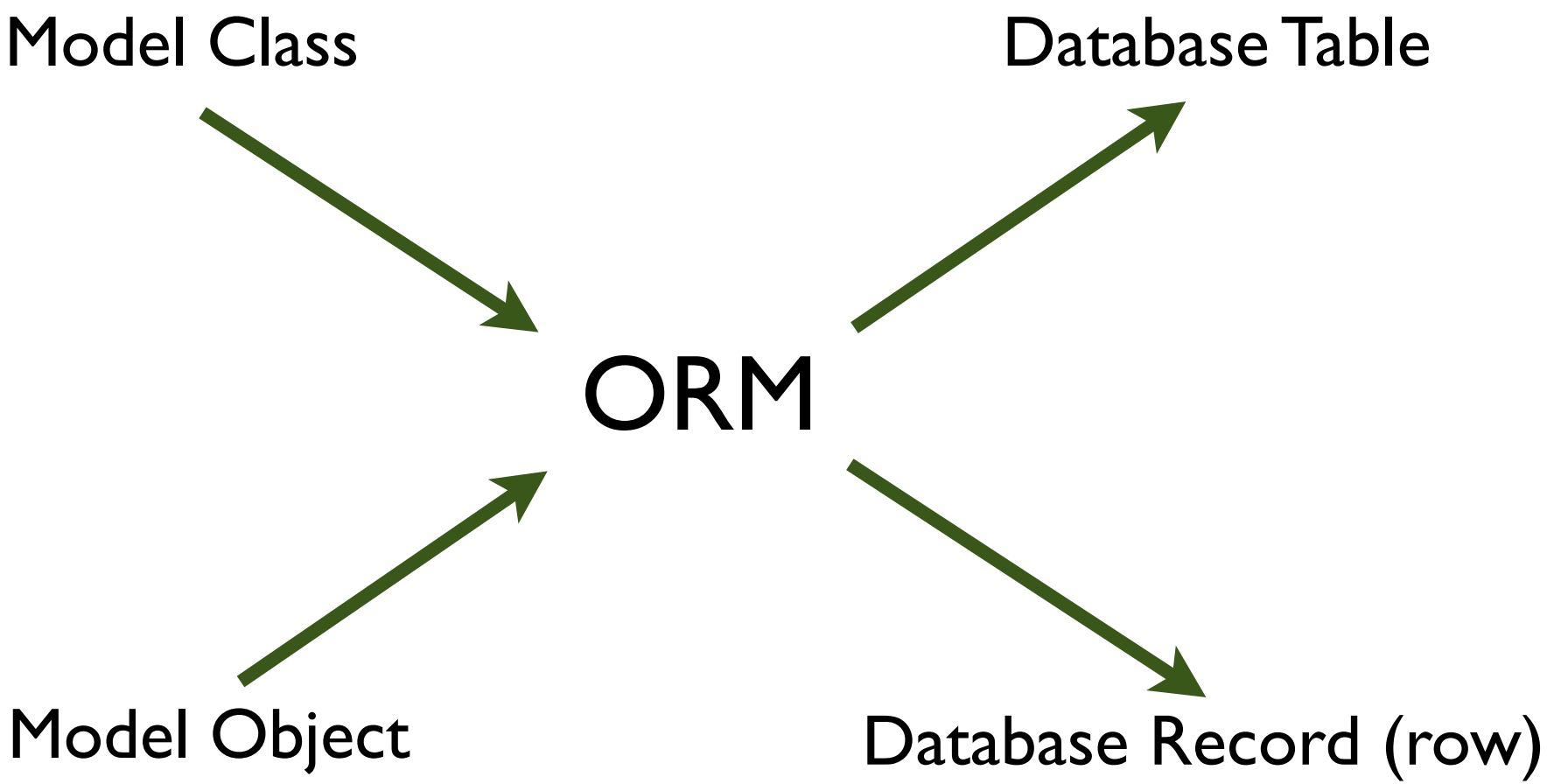
```
from flask import Flask
from flask.ext.sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:////tmp/test.db'
db = SQLAlchemy(app)


class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    email = db.Column(db.String(120), unique=True)

    def __init__(self, username, email):
        self.username = username
        self.email = email
```

Model Relationships

```
class Manufacturer(db.Model):
    # ...

class Car(db.Model):
    company = db.relationship('Manufacturer')
```



Models


Model Class
Database Table
ORM
Model Object
Database Record (row)

Flask Apps

The philosophy of modern web frameworks is "Don't Repeat Yourself" (DRY).

Flask is already good at supplying the DRY building blocks for low-level tasks, but what about high-level functionality?

User registration (Flask-Login)

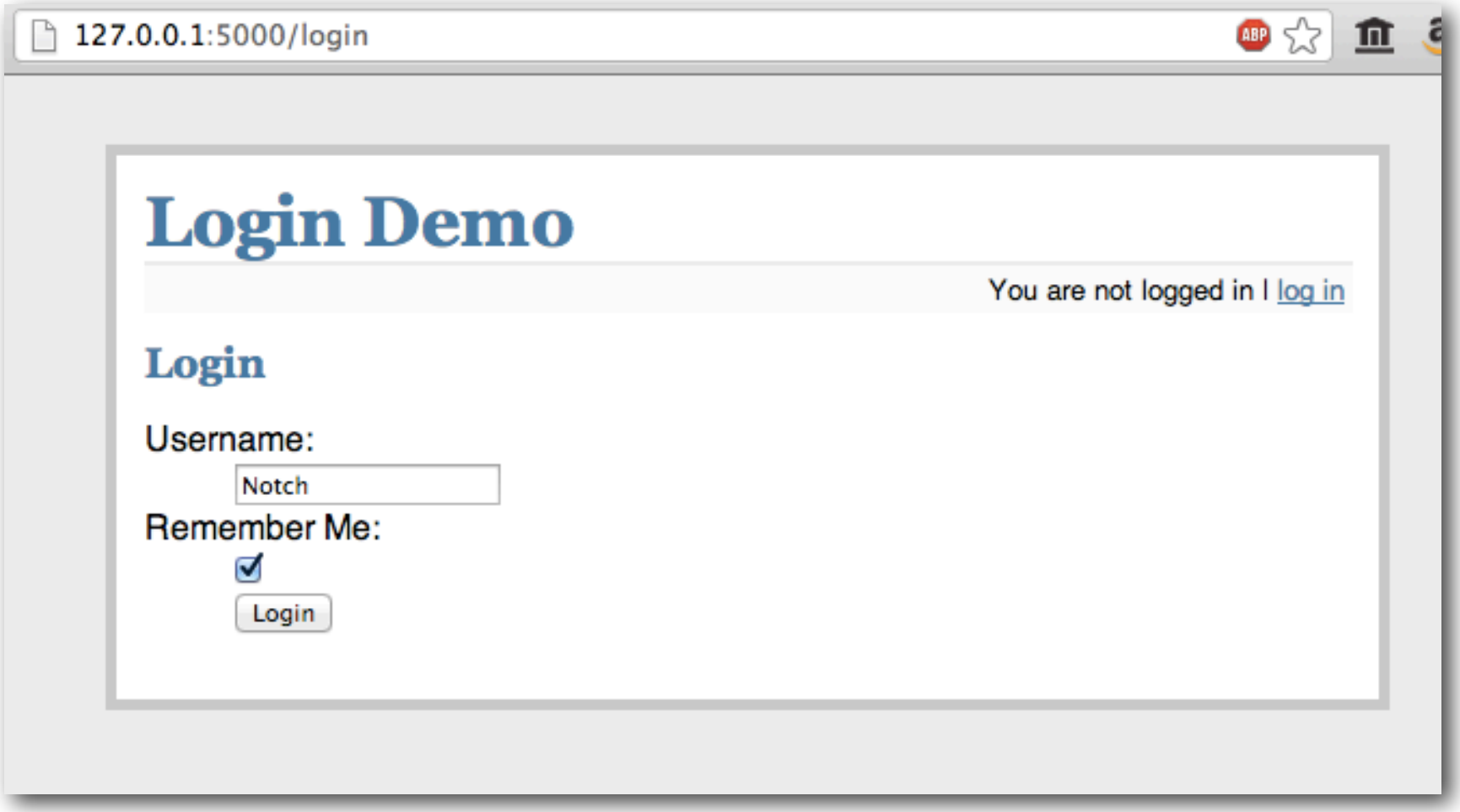
Sending Mail (Flask-Mail)

OpenID

Themes

The Flask community makes reusable apps to solve this problem. Plug it in and go.

pip install flask-login



flask.pocoo.org/extensions/

uploading files: http://flask.pocoo.org/docs/patterns/fileuploads/#uploading-files

Flask Extensions

Welcome to the Flask extensions registry. Here you can find a list of packages that extend Flask. This list is moderated and updated on a regular basis. If you want your package to show up here, follow the guide on creating extensions.

Flask-Admin

Flask extension module that provides an admin interface

Author:	Serge Koval
PyPI Page:	Flask-Admin
Documentation:	Read docs @ flask-admin.readthedocs.org
On Github:	mrjoes/flask-admin

Build something sciencey

How about a journal?

Google App Engine – Google Code

http://code.google.com/appengine/

google app engine

Apple Yahoo! Google Maps YouTube Wikipedia Popular News (430) Add to Boxee Note in Reader Scan for Privacy

Google App Engine – Google Code

profjsb@gmail.com | My favorites | English | Sign ou

Google code

Search

e.g. "templates" or "datastore"

Google App Engine

Home Docs FAQ Articles Blog Community Terms Download

Run your web apps on Google's infrastructure.

Easy to build, easy to maintain, easy to scale.

Google App Engine enables you to build and host web apps on the same systems that power Google applications. App Engine offers fast development and deployment; simple administration, with no need to worry about hardware, patches or backups; and effortless scalability. Discover why developers are choosing App Engine.

Google App Engine for Business

New!

Run your enterprise applications on Google's infrastructure.

Getting Started

1. Sign up for an App Engine account.
2. Download the App Engine SDK.
3. Read the Getting Started Guide.

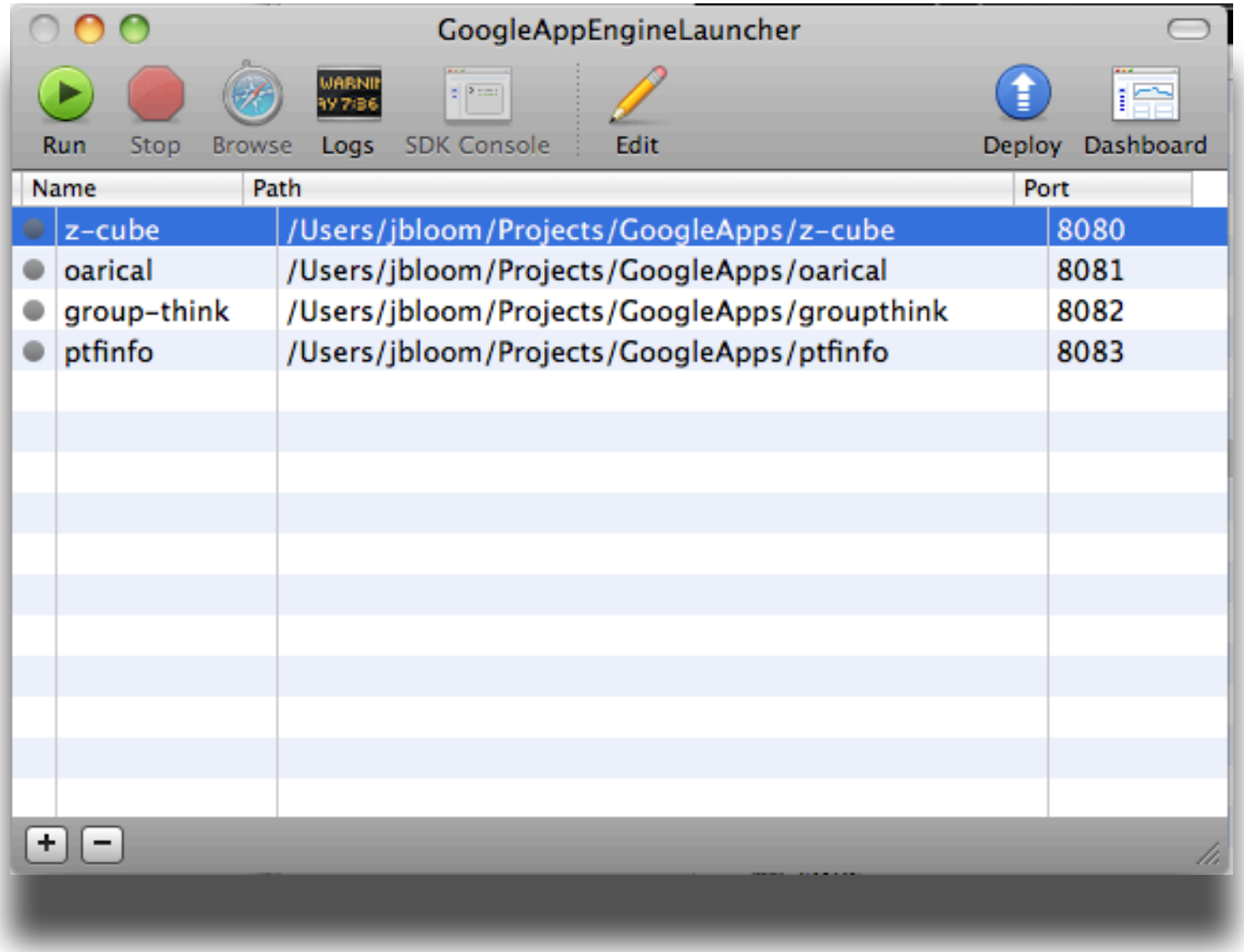
Watch and Learn

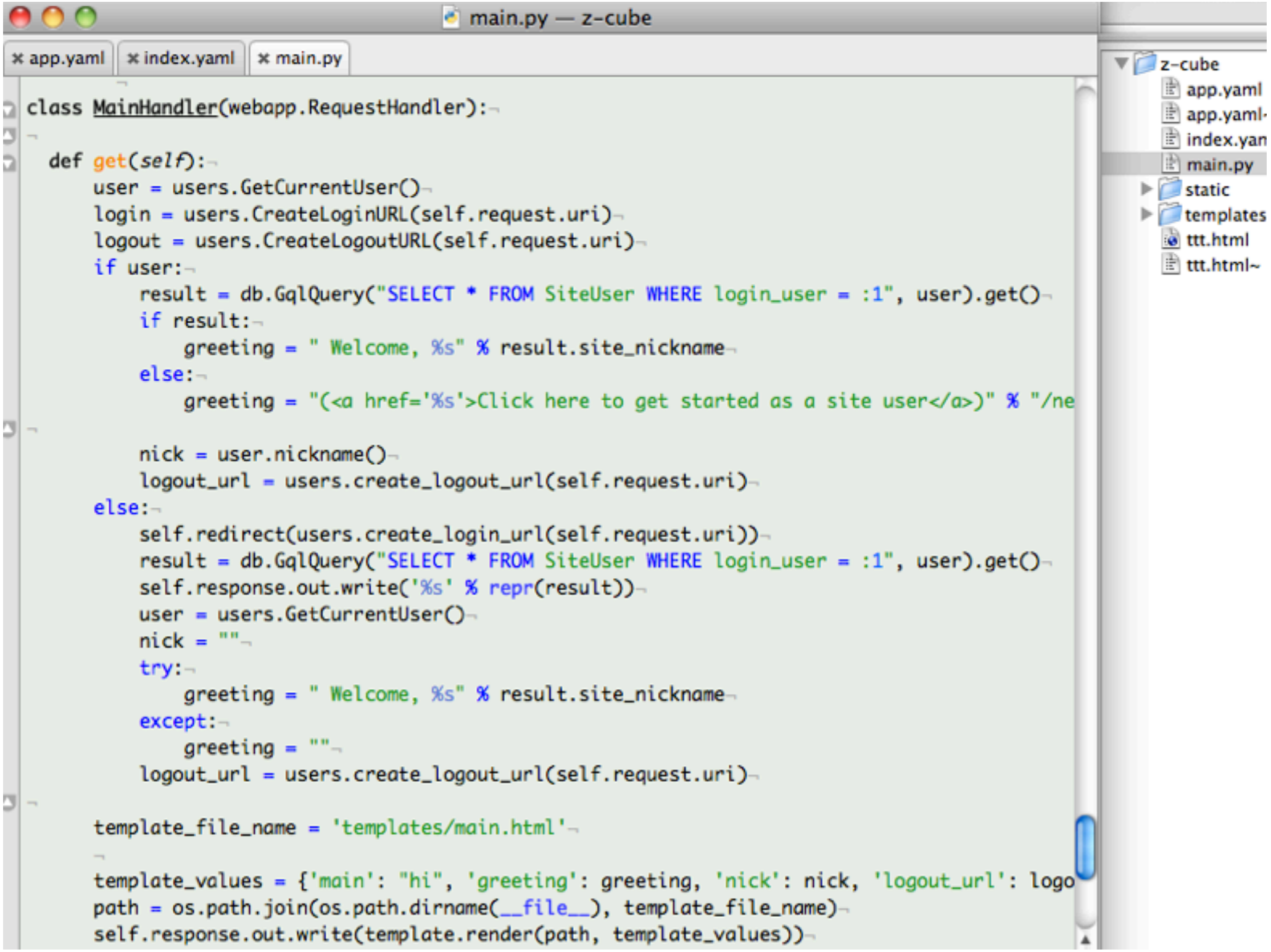
Developing and deploying on Google App Engine. Watch Now

links from AppEngine.reddit.com

New App Engine SDK 1.2.9 includes

SDK allows you to run a local version of your projects




main.py — z-cube
app.yaml
index.yaml
main.py
z-cube
app.yaml
app.yaml~
index.yaml
main.py
static
templates
ttt.html
ttt.html~
class MainHandler(webapp.RequestHandler):
def get(self):
user = users.GetCurrentUser()
login = users.CreateLoginURL(self.request.uri)
logout = users.CreateLogoutURL(self.request.uri)
if user:
result = db.GqlQuery("SELECT * FROM SiteUser WHERE login_user = :1", user).get()
if result:
greeting = " Welcome, %s" % result.site_nickname
else:
greeting = "(Click here to get started as a site user)" % "/ne
nick = user.nickname()
logout_url = users.create_logout_url(self.request.uri)
else:
self.redirect(users.create_login_url(self.request.uri))
result = db.GqlQuery("SELECT * FROM SiteUser WHERE login_user = :1", user).get()
self.response.out.write('%s' % repr(result))
user = users.GetCurrentUser()
nick = ""
try:
greeting = " Welcome, %s" % result.site_nickname
except:
greeting = ""
logout_url = users.create_logout_url(self.request.uri)
template_file_name = 'templates/main.html'
template_values = {'main': "hi", 'greeting': greeting, 'nick': nick, 'logout_url': logo
path = os.path.join(os.path.dirname(__file__), template_file_name)
self.response.out.write(template.render(path, template_values))

online management...