Web Development with Python

AY250 Fall 2016

```
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

I SHOULD BUILD MY ENTIRE WEBSTACK WITH PYTHON
megenerator.net

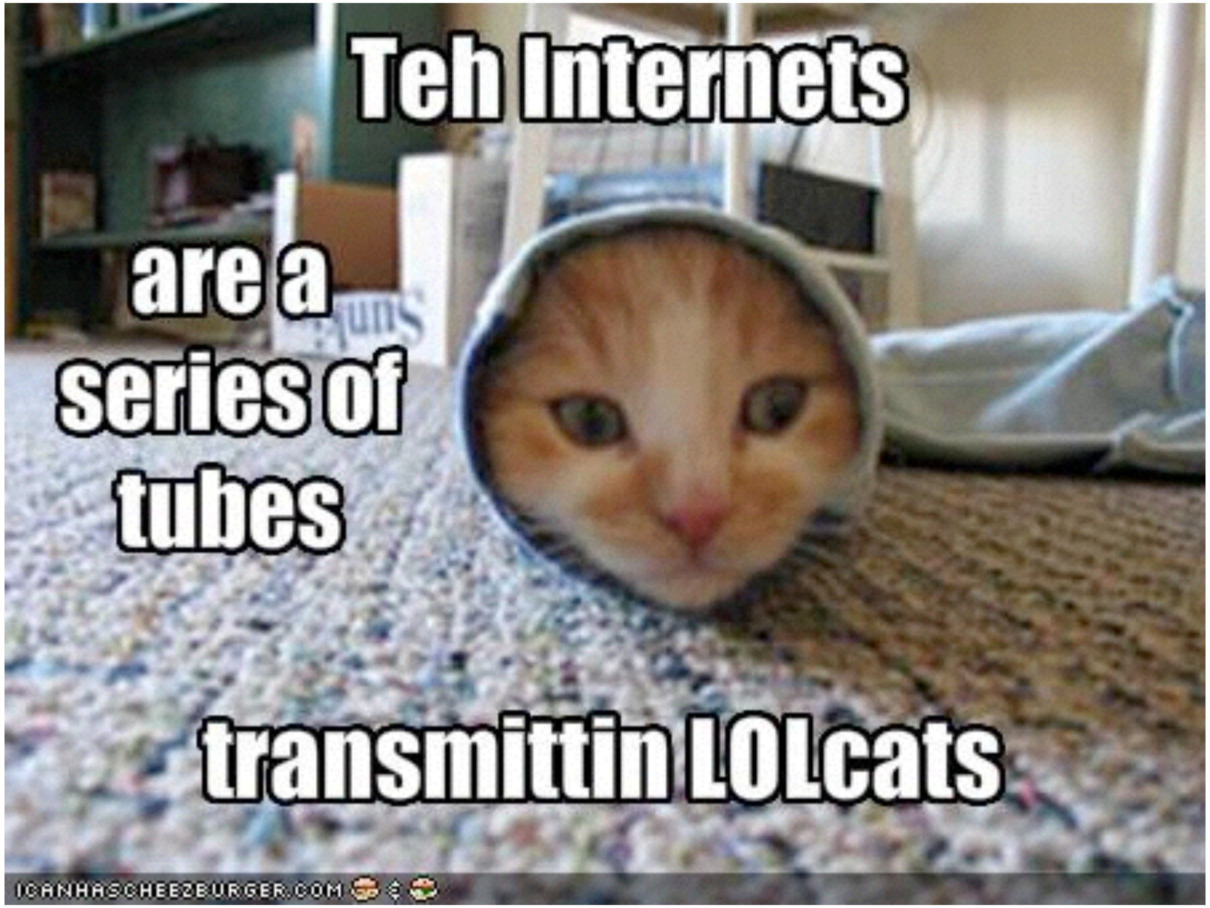
Overview of Today's Lecture

The web paradigm

Using Python for web supremacy

- Basic Python servers
- Frameworks and using Flask
- Platforms: e.g. Google App Engine

Believe it or not



the internet was not built for this.

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

Request and Response

User knocks on the server’s door

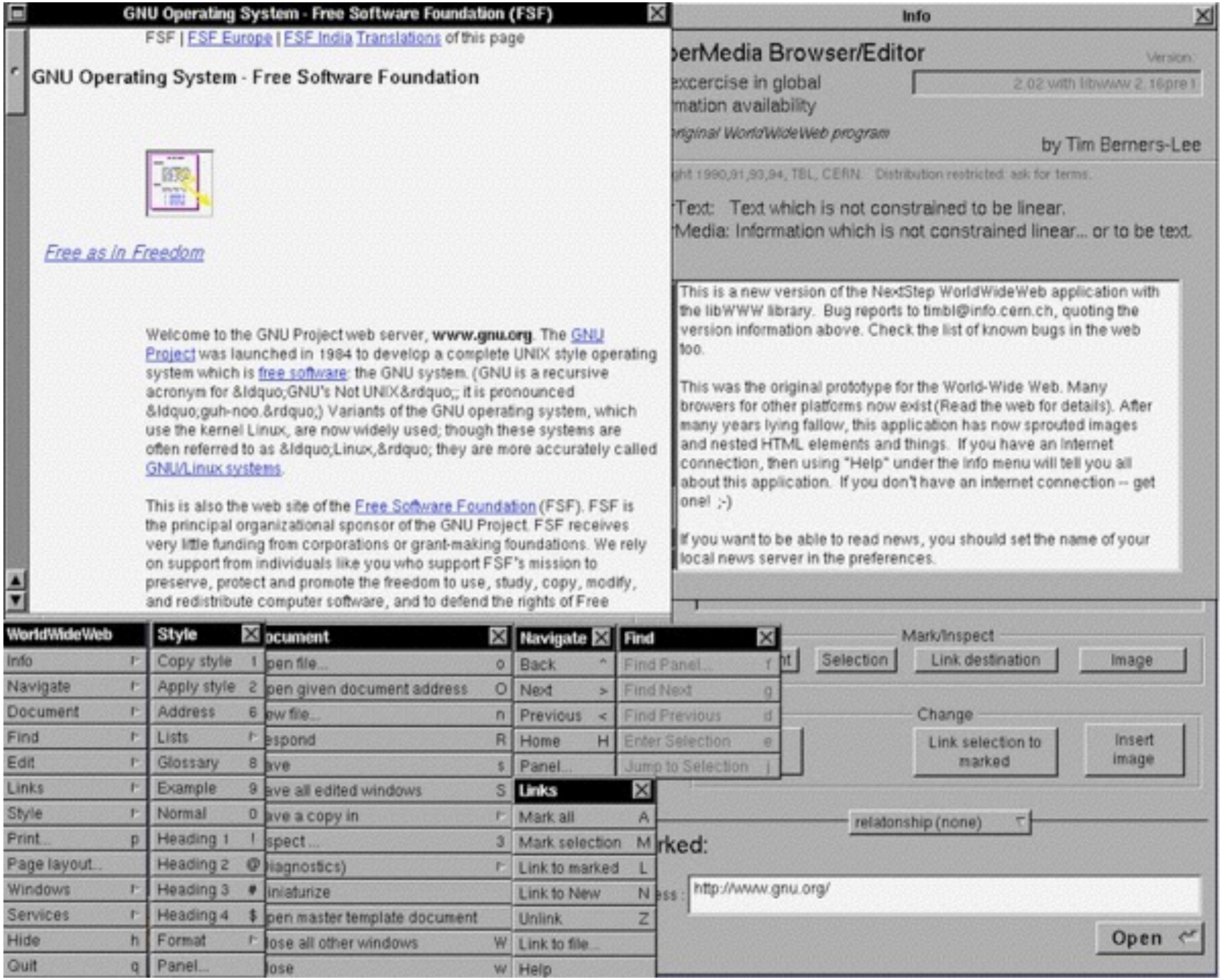
(request)

Server thinks about it

Server sends data back to the user

(response)

The browser model



GNU Operating System - Free Software Foundation (FSF)

FSF | FSF Europe | FSF India Translations of this page

GNU Operating System - Free Software Foundation

Free as in Freedom

Welcome to the GNU Project web server, **www.gnu.org**. The GNU Project was launched in 1984 to develop a complete UNIX style operating system which is free software: the GNU system. (GNU is a recursive acronym for &ldquo;GNU's Not UNIX&rdquo;; it is pronounced &ldquo;guh-noo.&rdquo;) Variants of the GNU operating system, which use the kernel Linux, are now widely used; though these systems are often referred to as &ldquo;Linux,&rdquo; they are more accurately called GNU/Linux systems.

This is also the web site of the Free Software Foundation (FSF). FSF is the principal organizational sponsor of the GNU Project. FSF receives very little funding from corporations or grant-making foundations. We rely on support from individuals like you who support FSF's mission to preserve, protect and promote the freedom to use, study, copy, modify, and redistribute computer software, and to defend the rights of Free

Info

perMedia Browser/Editor

Version: 2.02 with libwww 2.16pre1

xcercise in global
mation availability

riginal WorldWideWeb program

by Tim Berners-Lee

ght 1990,91,93,94, TBL, CERN. Distribution restricted: ask for terms.

Text: Text which is not constrained to be linear.
Media: Information which is not constrained linear... or to be text.

This is a new version of the NextStep WorldWideWeb application with the libWWW library. Bug reports to timbl@info.cern.ch, quoting the version information above. Check the list of known bugs in the web too.

This was the original prototype for the World-Wide Web. Many browers for other platforms now exist (Read the web for details). After many years lying fallow, this application has now sprouted images and nested HTML elements and things. If you have an internet connection, then using "Help" under the Info menu will tell you all about this application. If you don't have an internet connection -- get one! ;-)

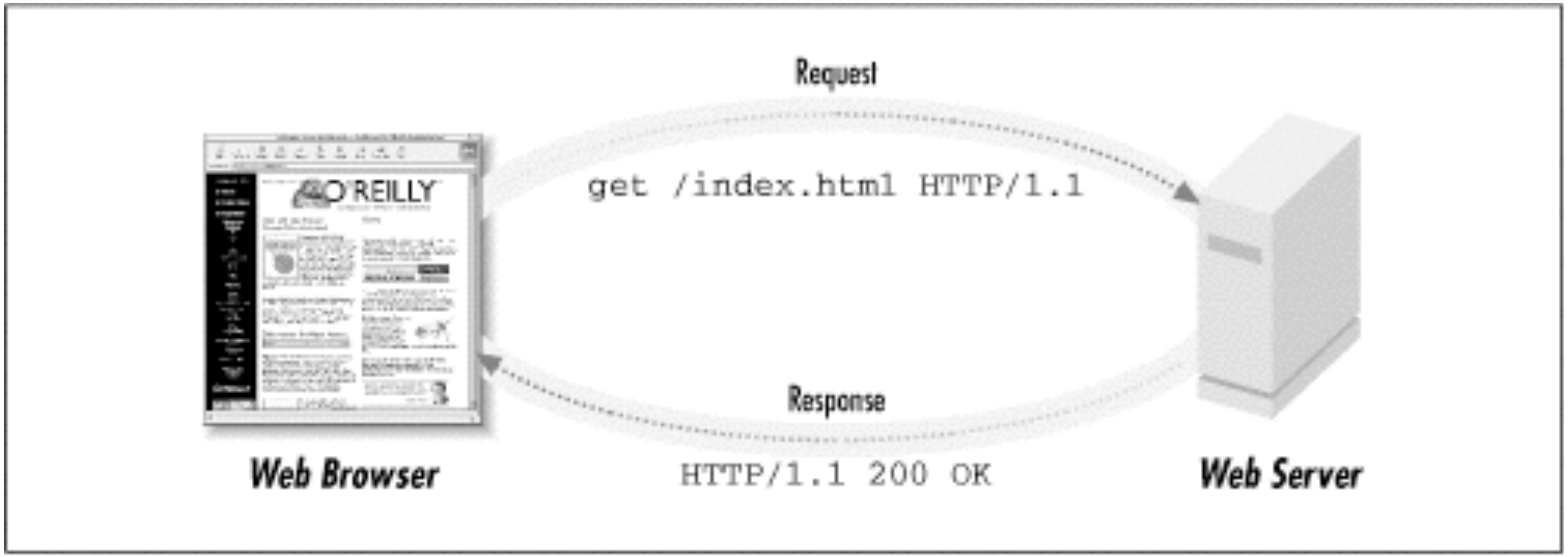
If you want to be able to read news, you should set the name of your local news server in the preferences.

rked:

ess: http://www.gnu.org/

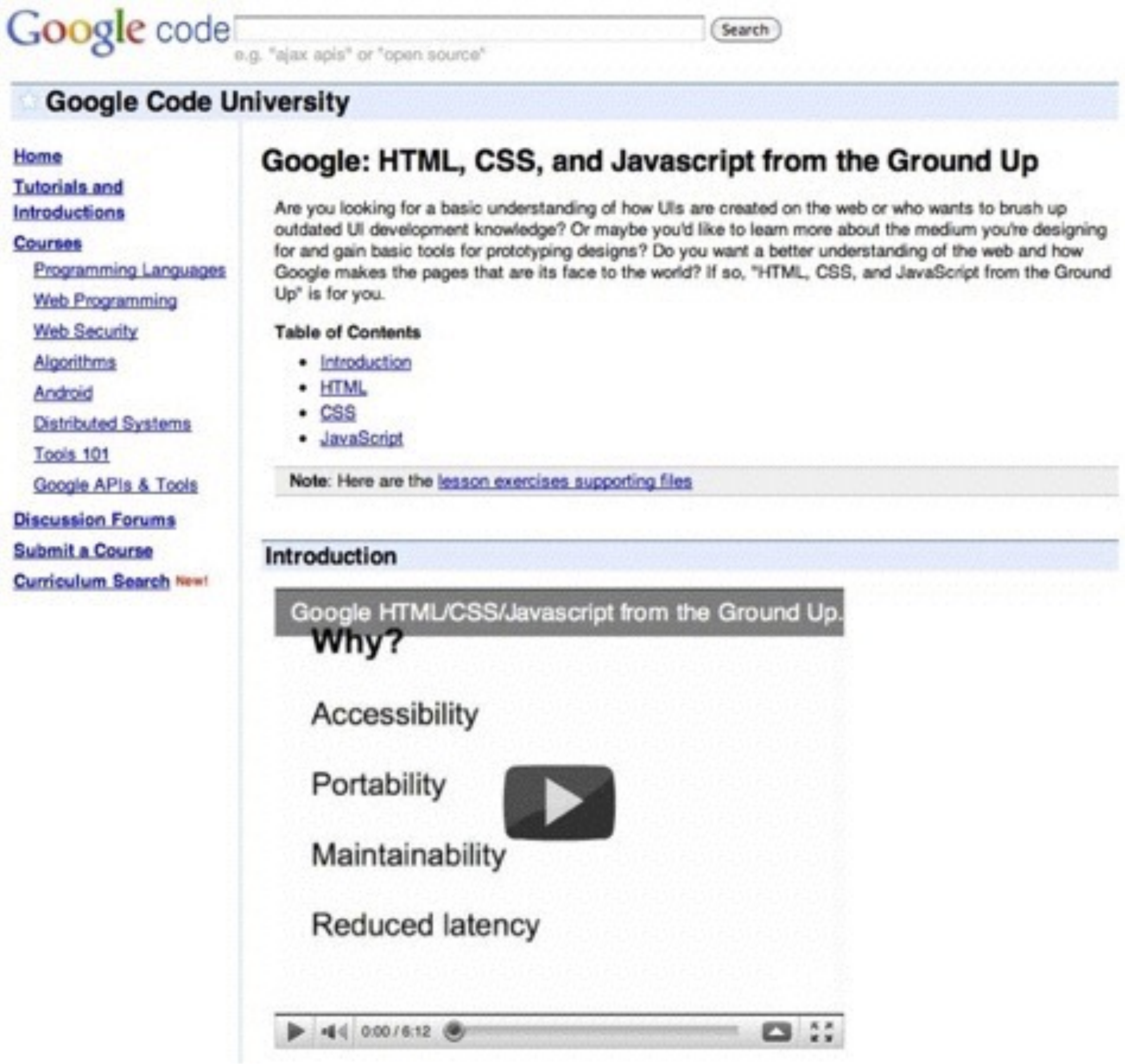
http://en.wikipedia.org/wiki/Web_browser

Request and Response



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

Browser stuff



Google Code University

Google: HTML, CSS, and Javascript from the Ground Up

Are you looking for a basic understanding of how UIs are created on the web or who wants to brush up outdated UI development knowledge? Or maybe you'd like to learn more about the medium you're designing for and gain basic tools for prototyping designs? Do you want a better understanding of the web and how Google makes the pages that are its face to the world? If so, "HTML, CSS, and JavaScript from the Ground Up" is for you.

Table of Contents

- Introduction
- HTML
- CSS
- JavaScript

Note: Here are the lesson exercises supporting files

Introduction

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

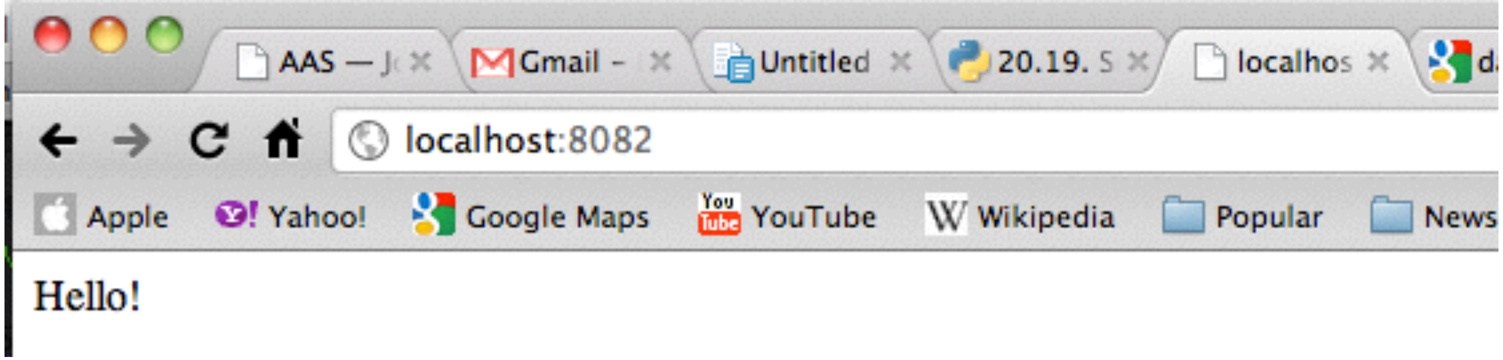
Recall the XML/RPC Server...

a webserver just responds to a different request protocol

```
import http.server
class myresponse(http.server.SimpleHTTPRequestHandler):
   def do_GET(s):
       s.wfile.write("<body>Hello!</body>".encode("UTF-8"))

httpd = http.server.HTTPServer(("localhost", 8084), myresponse)
httpd.serve_forever()
```

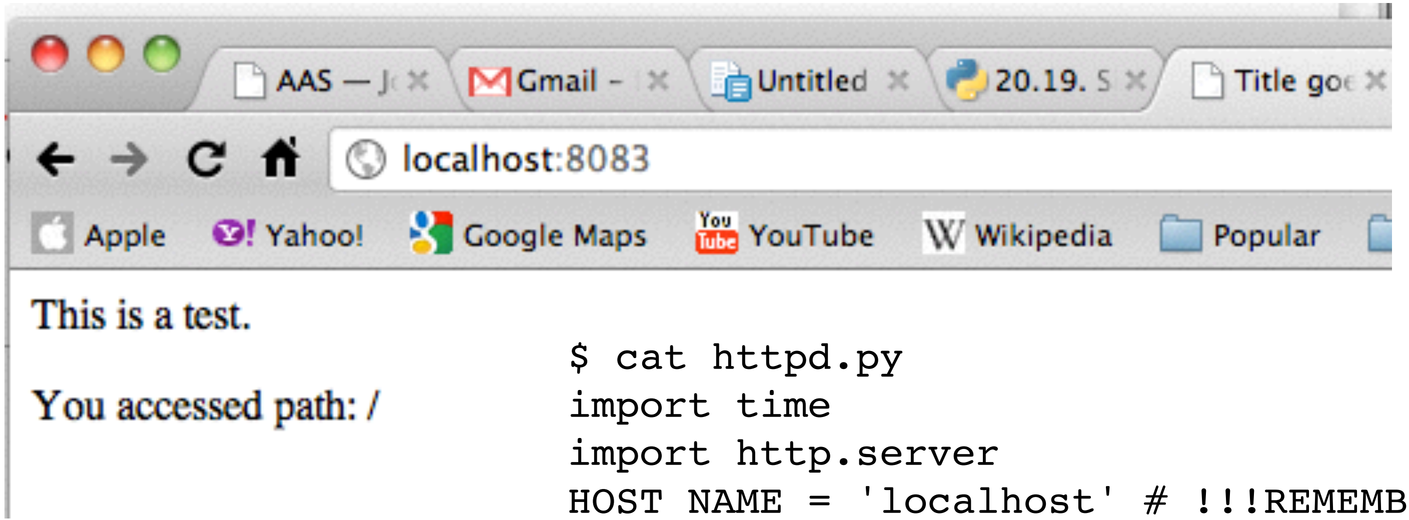
`wfile`: output stream file

note: HTTPServer is not threaded



Hello!

```
$ python httpd.py
127.0.0.1 - - [06/Oct/2016 14:15:06] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/Oct/2016 14:15:08] "GET /favicon.ico HTTP/1.1" 200 -
```



This is a test.

You accessed path: /

```
$ cat httpd.py
import time
import http.server
HOST_NAME = 'localhost' # !!!REMEMBER TO CHANGE THIS!!!
PORT_NUMBER = 8090 # Maybe set this to 9000.
def s2b(s):
    return s.encode("UTF-8")
class MyHandler(http.server.SimpleHTTPRequestHandler):
    def do_HEAD(s):
        s.send_response(200)
        s.send_header(s2b("Content-type"), s2b("text/ht
        s.end_headers()

    def do_GET(s):
        """Respond to a GET request."""
        s.send_response(200)
        s.send_header(s2b("Content-type"), s2b("text/ht
```

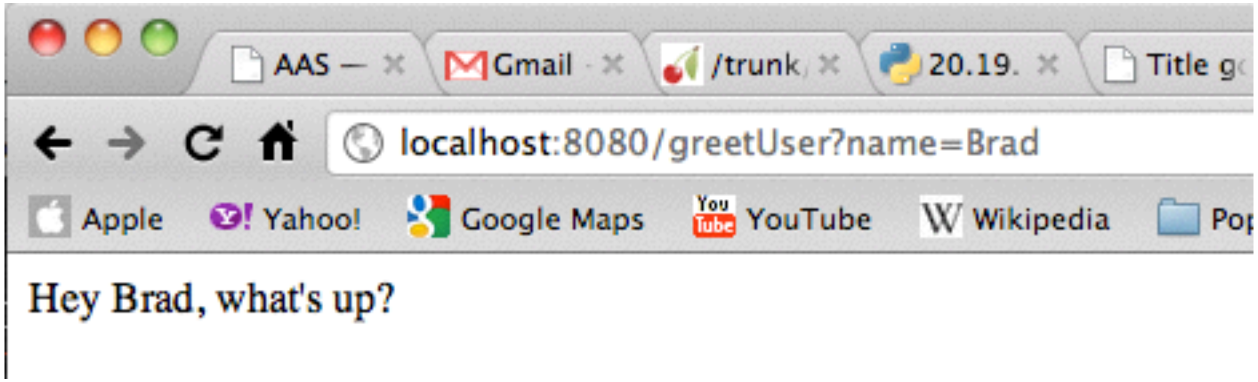
Simple Servers

CherryPy : `pip install cherrypy`

```
import cherrypy

PORTNUM = 8093

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:{}/greetUser?name=Josh</i>'.format(PORTNUM)
    greetUser.exposed = True

cherrypy.config.update({"server.socket_port": PORTNUM})

cherrypy.quickstart(WelcomePage())
```

file: cp1.py

here, name is a variable of the GET request



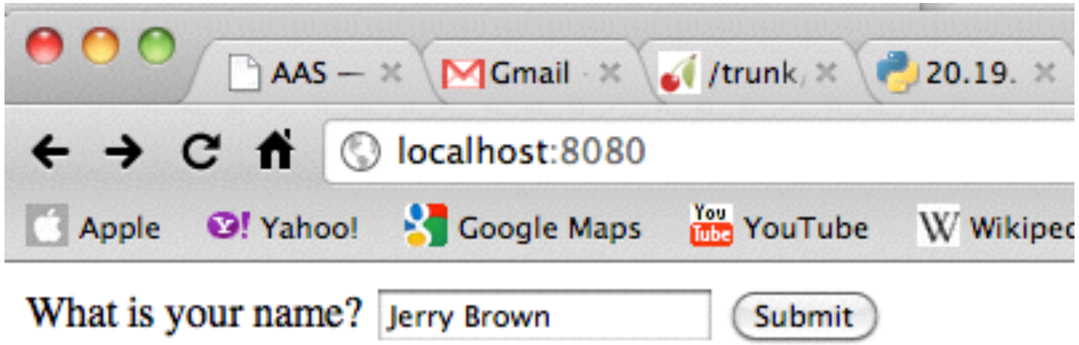
localhost:8080/greetUser?name=Brad

Hey Brad, what's up?

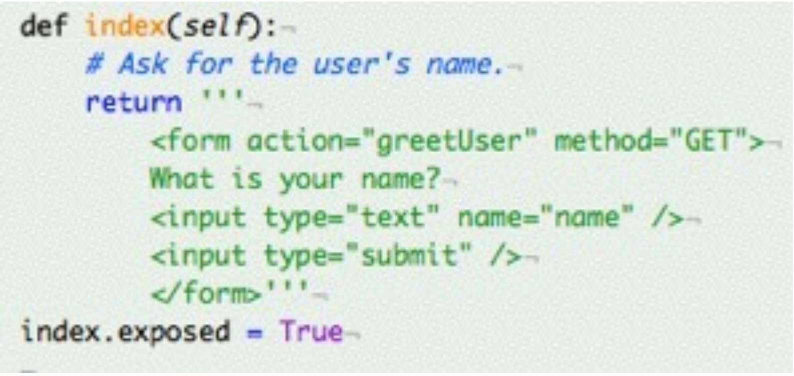
Simple Servers

CherryPy

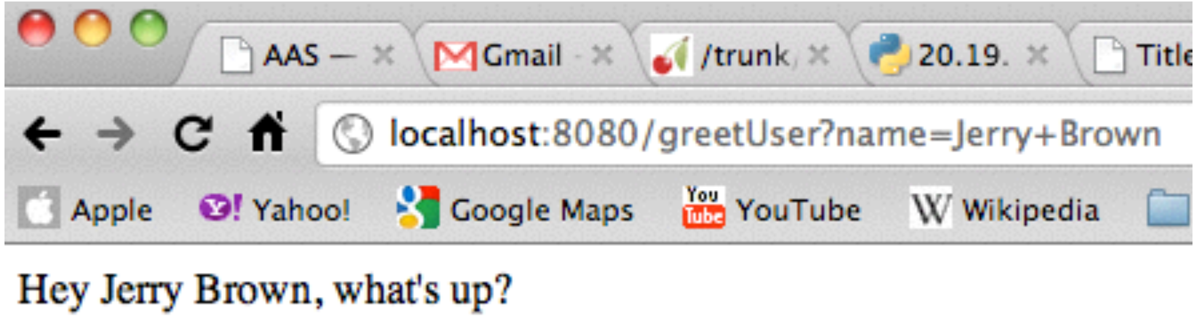
```
>>> run cp2
```



What is your name? Jerry Brown

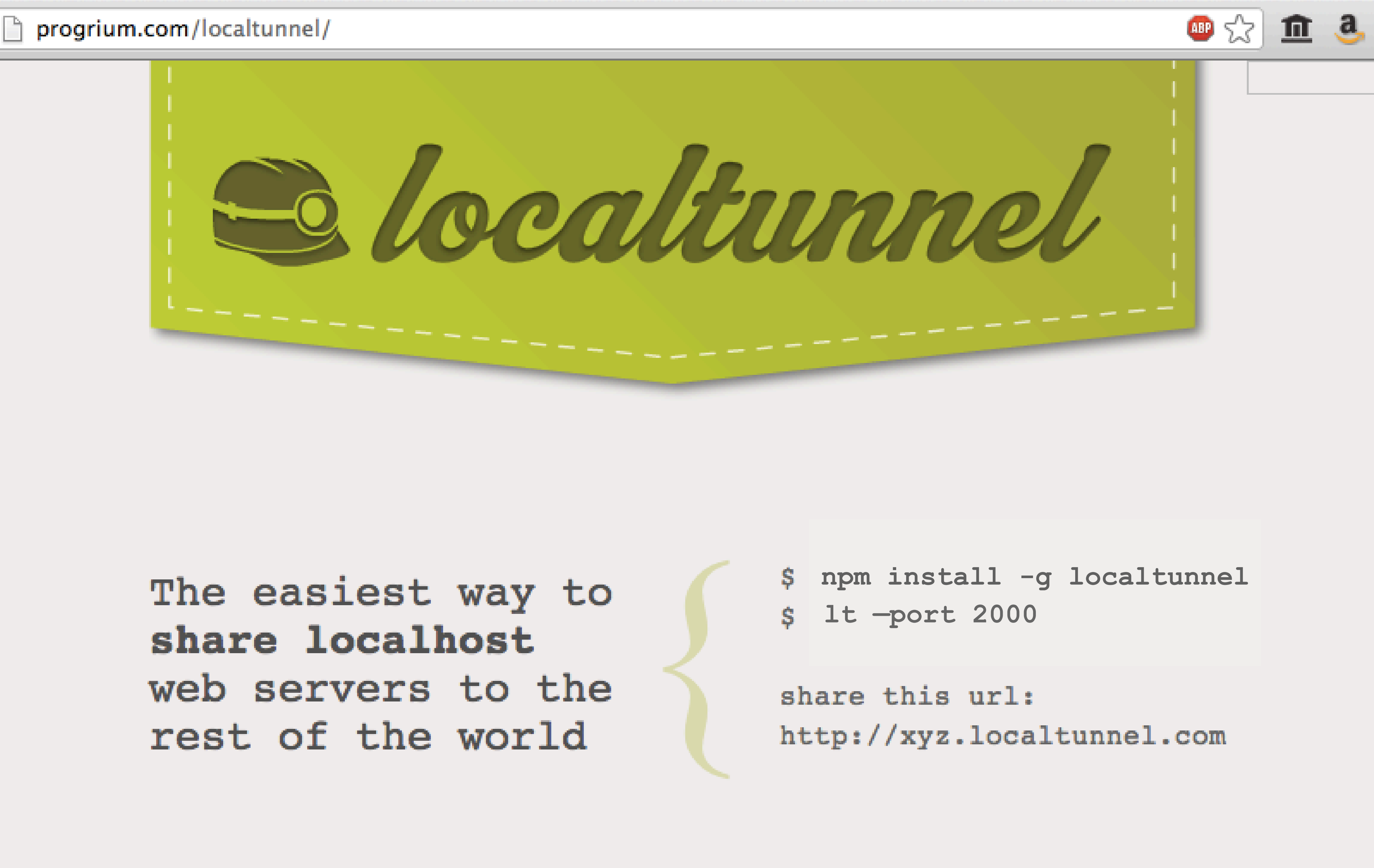


```
def index(self):
    # Ask for the user's name.
    return '''
        <form action="greetUser" method="GET">
        What is your name?
        <input type="text" name="name" />
        <input type="submit" />
        </form>'''
index.exposed = True
```



Hey Jerry Brown, what's up?

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



progrium.com/localtunnel/

The easiest way to **share localhost** web servers to the rest of the world

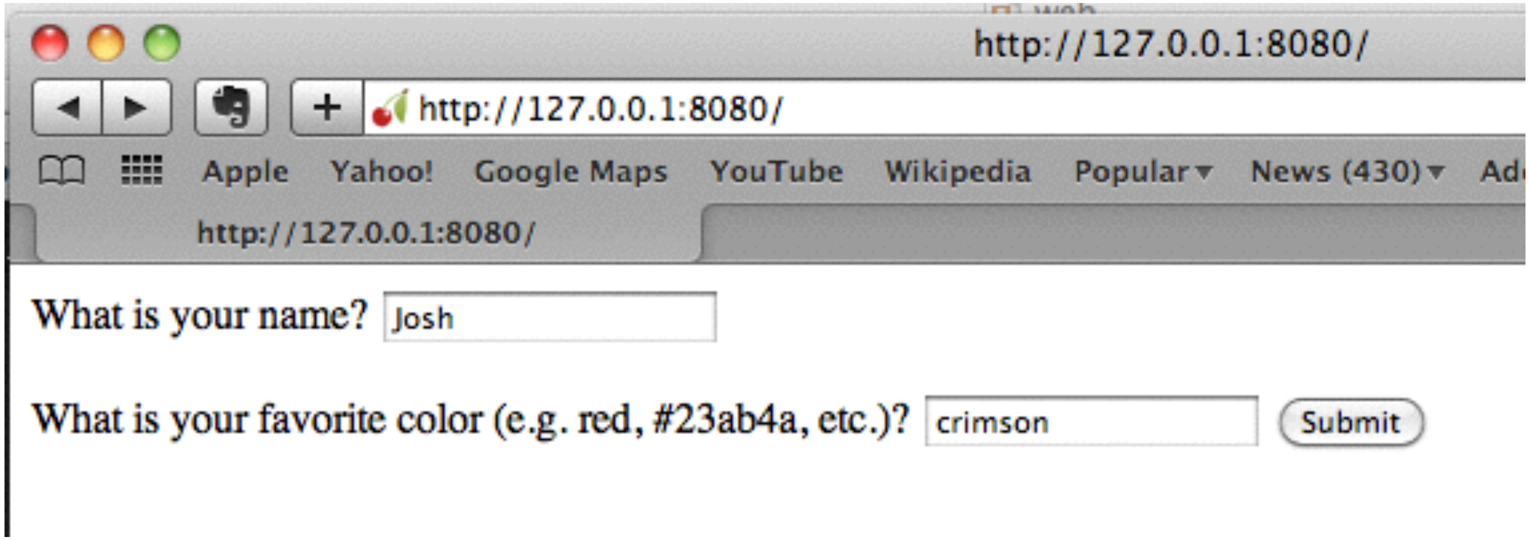
```
$ npm install -g localtunnel
$ lt —port 2000
```

share this url:
http://xyz.localtunnel.com

Small exercise:

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

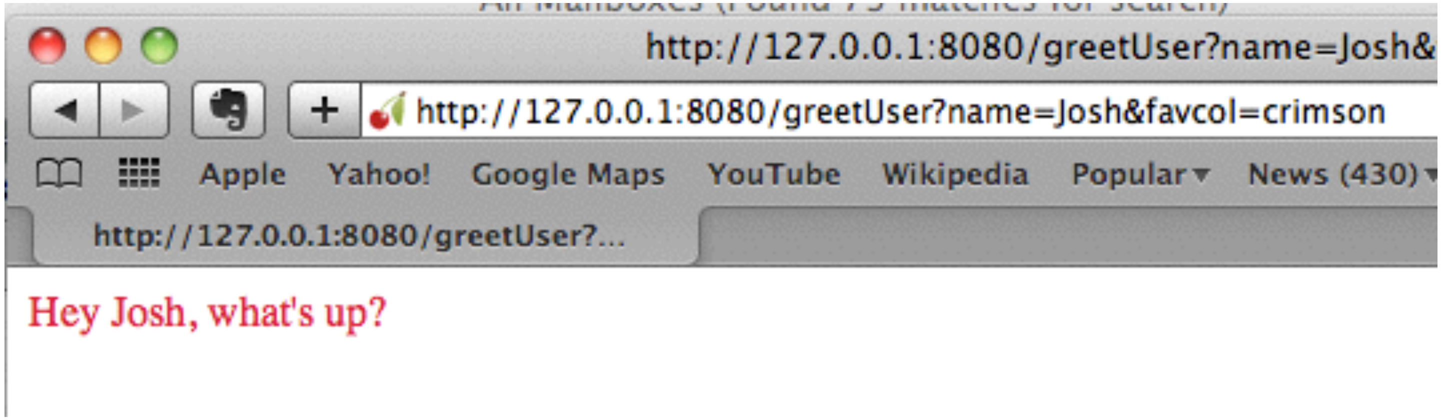
```
<font color="red">output</font>
```



http://127.0.0.1:8080/

What is your name? Josh

What is your favorite color (e.g. red, #23ab4a, etc.)? crimson



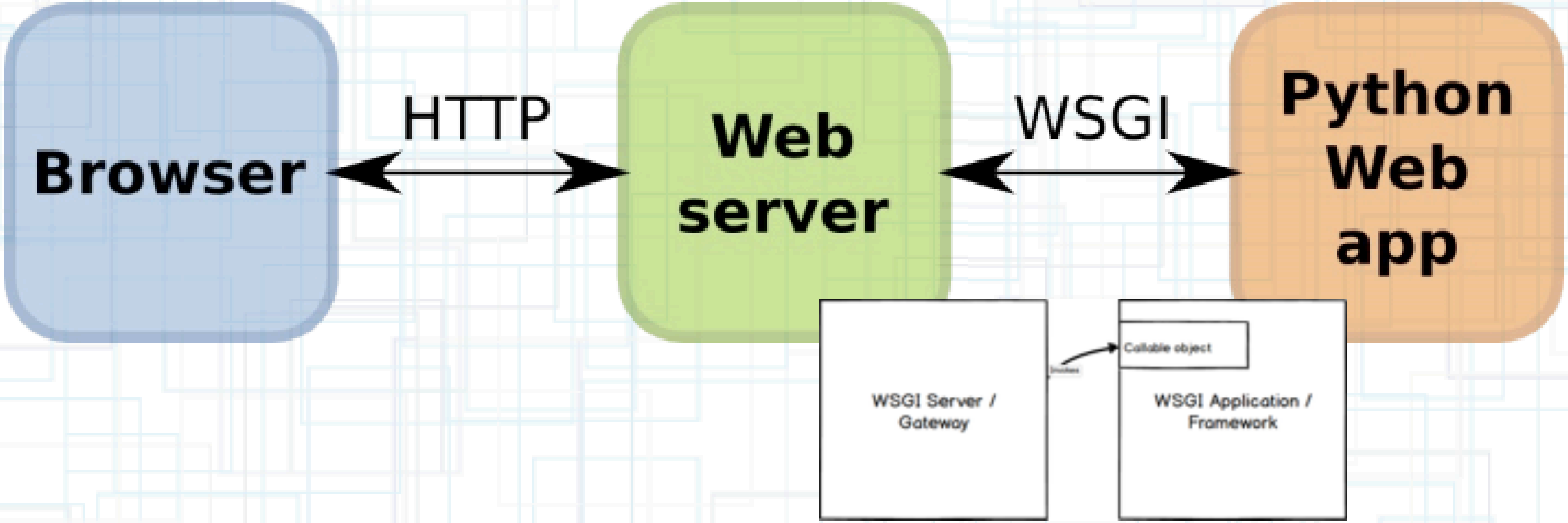
http://127.0.0.1:8080/greetUser?name=Josh&favcol=crimson

Hey Josh, what's up?

http://127.0.0.1:8080/greetUser?name=Josh&
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
Popular
News (430)
http://127.0.0.1:8080/greetUser?...

What's WSGI?

"simple and universal interface between web servers and web applications or frameworks for the Python programming language"



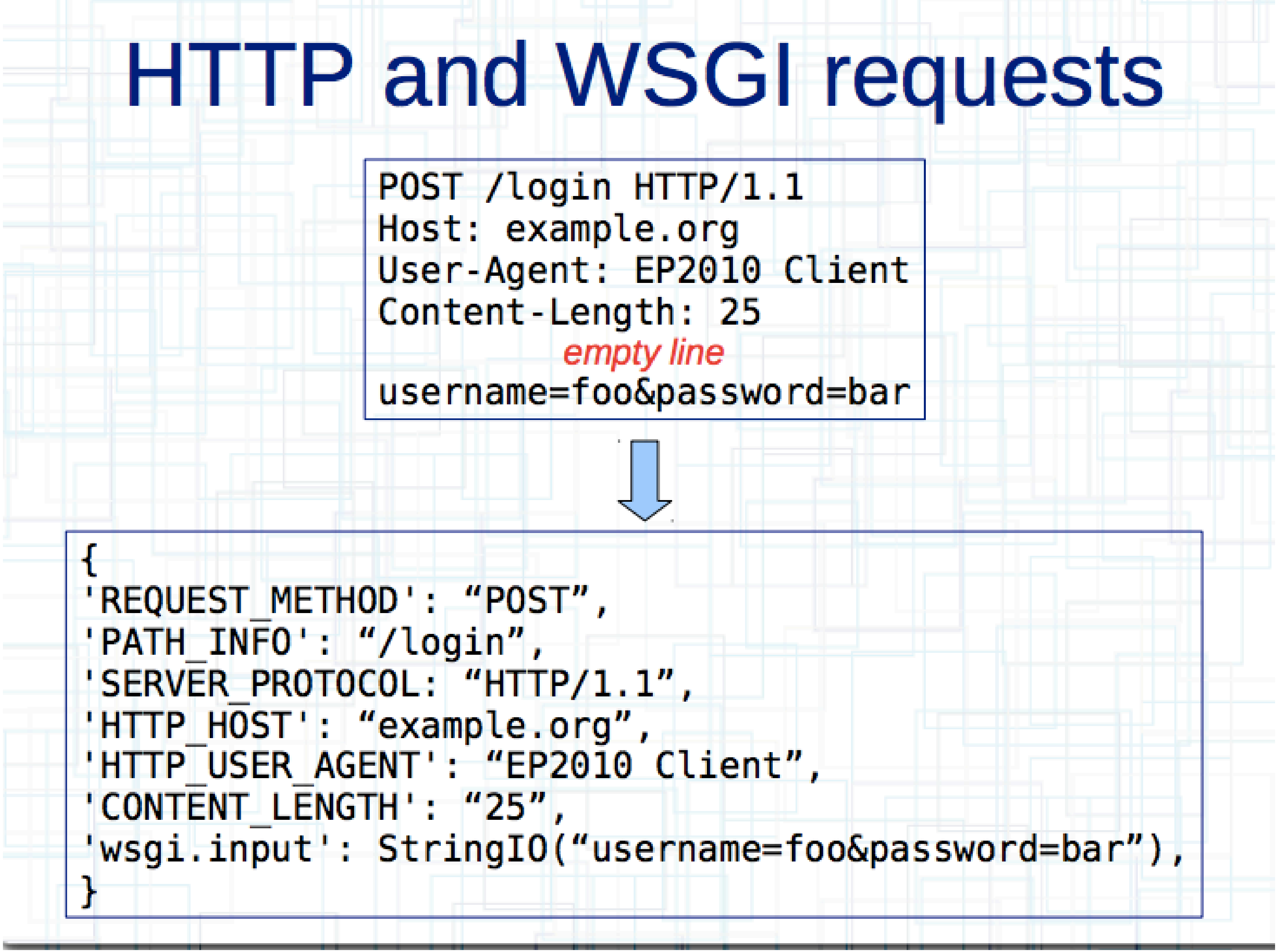
HTTP = HyperText Transfer Protocol

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

https://www.fullstackpython.com/wsgi-servers.html



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:

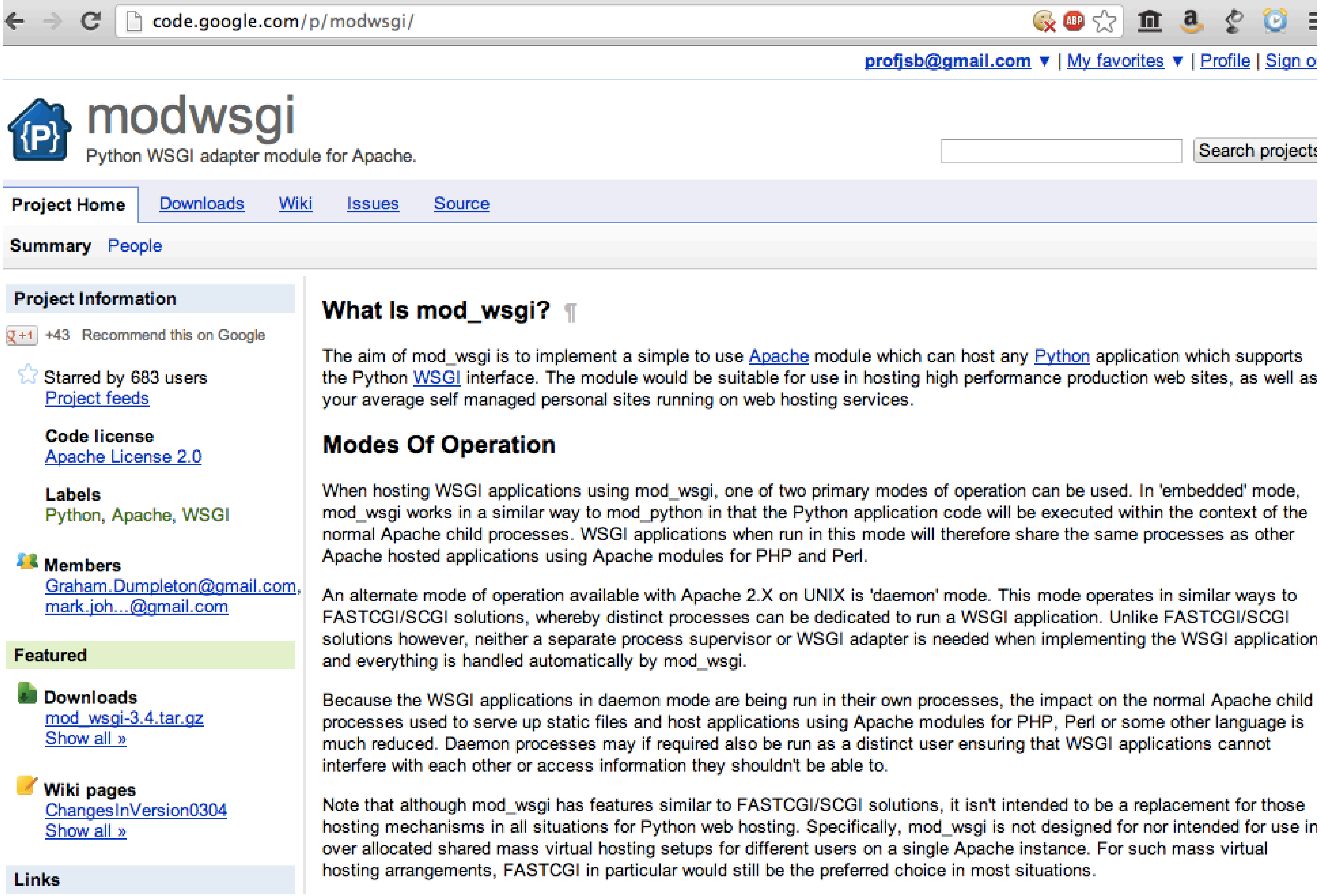
- speed, performance (multi-threaded, written in a compiled language like C)
- extensibility (written in pure Python)
- ease of development (code reloading, extra debugging features)
- adapters for a larger server platform (e.g. mod_wsgi for Apache, or the WSGI adapter for Google App Engine)

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

https://wsgi.readthedocs.io/en/latest/

https://bitbucket.org/lost_theory/wsgitalk

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


code.google.com/p/modwsgi/
profjsb@gmail.com ▼ | My favorites ▼ | Profile | Sign o
Search projects
Project Home
Downloads
Wiki
Issues
Source
Summary
People

modwsgi

Python WSGI adapter module for Apache.

Project Information

+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.

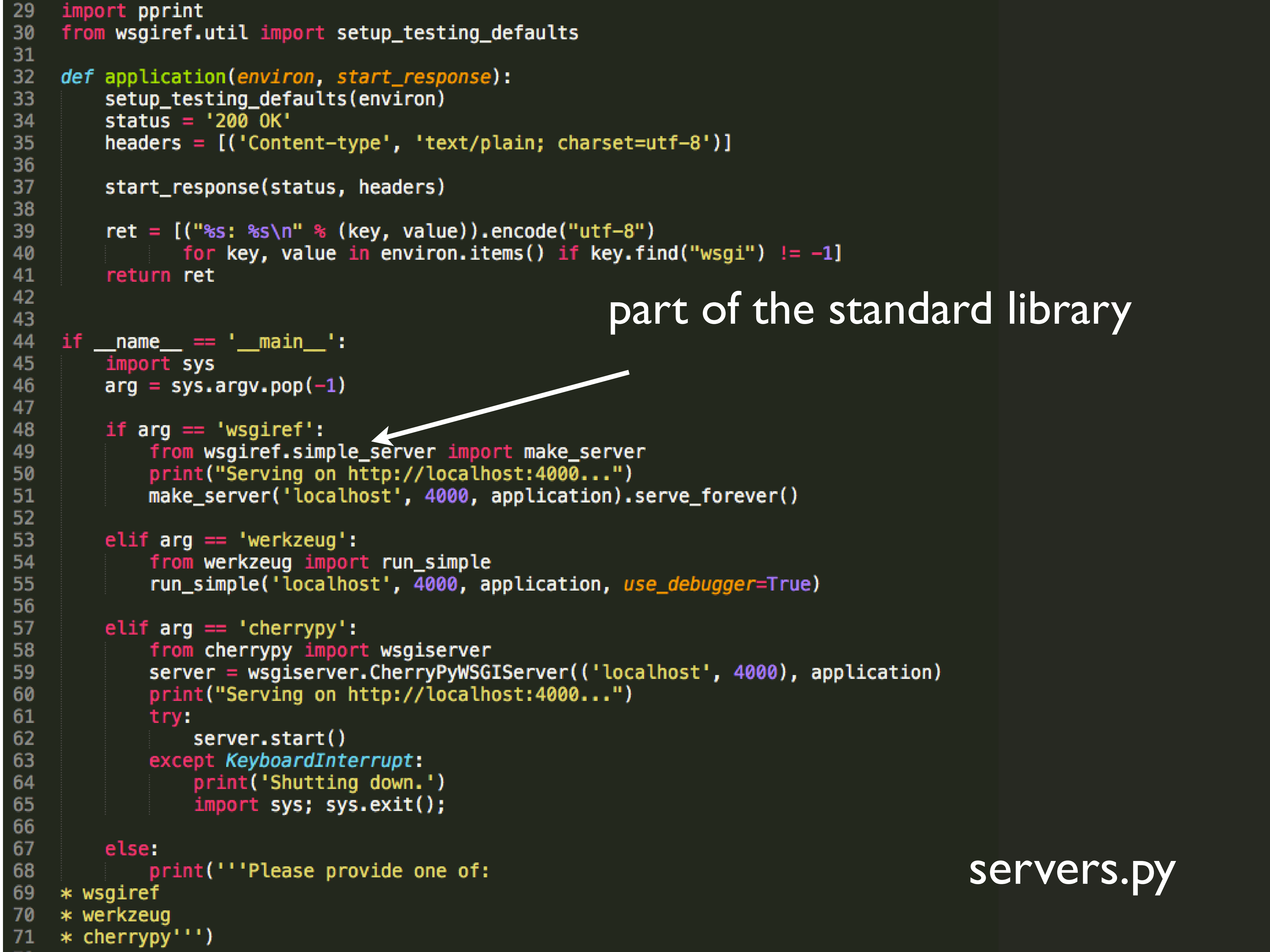
Modes Of Operation

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.



```python
import pprint
from wsgiref.util import setup_testing_defaults

def application(environ, start_response):
    setup_testing_defaults(environ)
    status = '200 OK'
    headers = [('Content-type', 'text/plain; charset=utf-8')]

    start_response(status, headers)

    ret = [("%s: %s\n" % (key, value)).encode("utf-8")
           for key, value in environ.items() if key.find("wsgi") != -1]
    return ret


if __name__ == '__main__':
    import sys
    arg = sys.argv.pop(-1)

    if arg == 'wsgiref':
        from wsgiref.simple_server import make_server
        print("Serving on http://localhost:4000...")
        make_server('localhost', 4000, application).serve_forever()

    elif arg == 'werkzeug':
        from werkzeug import run_simple
        run_simple('localhost', 4000, application, use_debugger=True)

    elif arg == 'cherrypy':
        from cherrypy import wsgiserver
        server = wsgiserver.CherryPyWSGIServer(('localhost', 4000), application)
        print("Serving on http://localhost:4000...")
        try:
            server.start()
        except KeyboardInterrupt:
            print('Shutting down.')
            import sys; sys.exit();

    else:
        print('''Please provide one of:
* wsgiref
* werkzeug
* cherrypy''')
```

part of the standard library

servers.py

Frameworks

Don’t spend time writing code for common tasks.

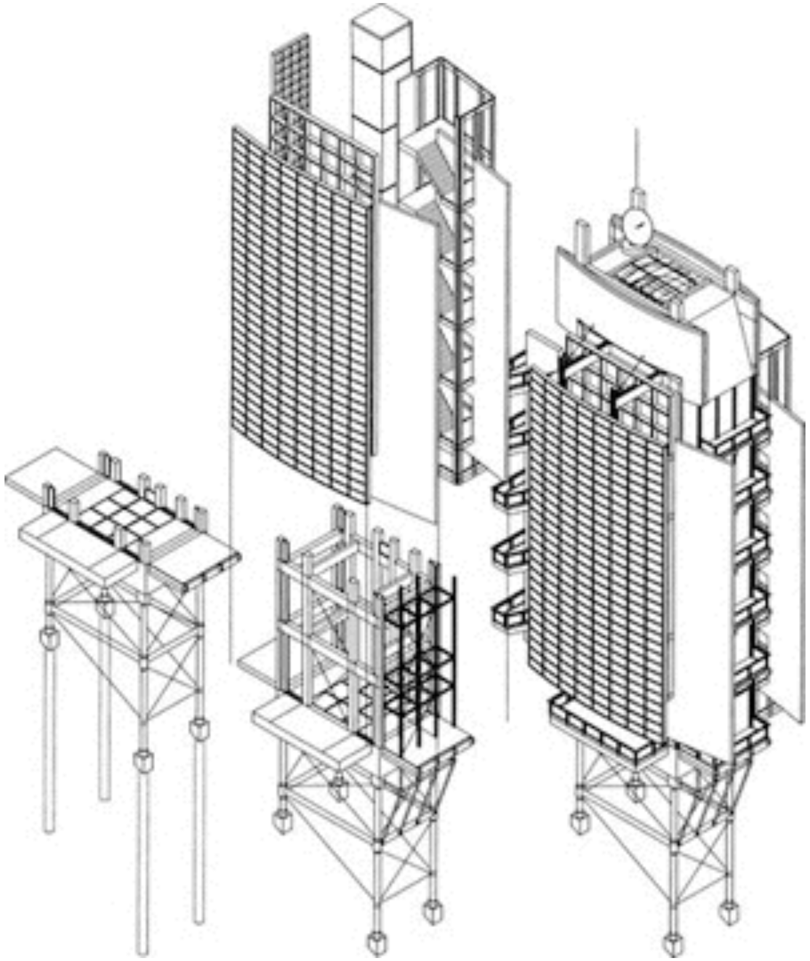
Query database B for activity of user 3...

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

http://flask.pocoo.org/

conda install flask

Hello World

fhello.py

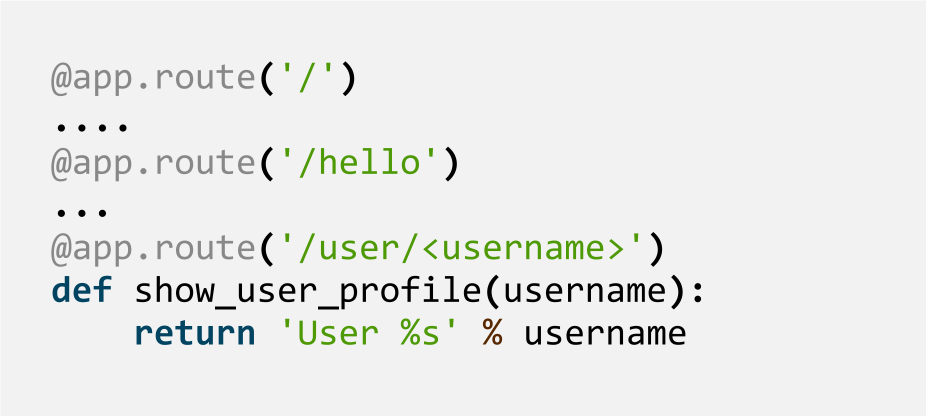
```
from flask import Flask
app = Flask(__name__)

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

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

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

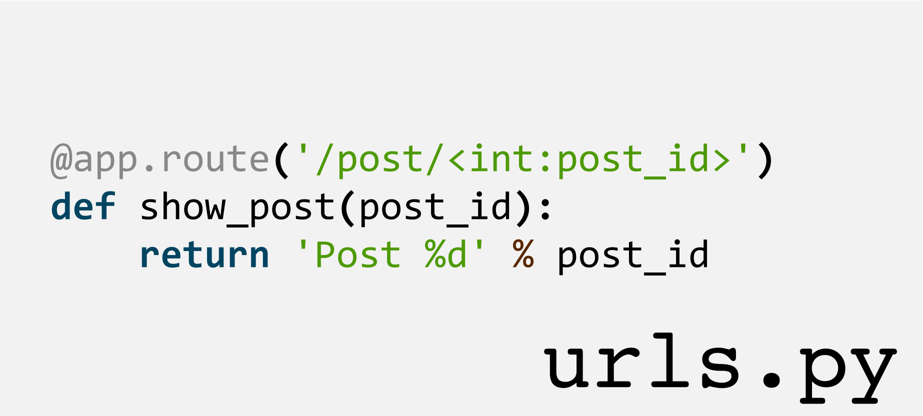
try: thello.py

URL Route Registration

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



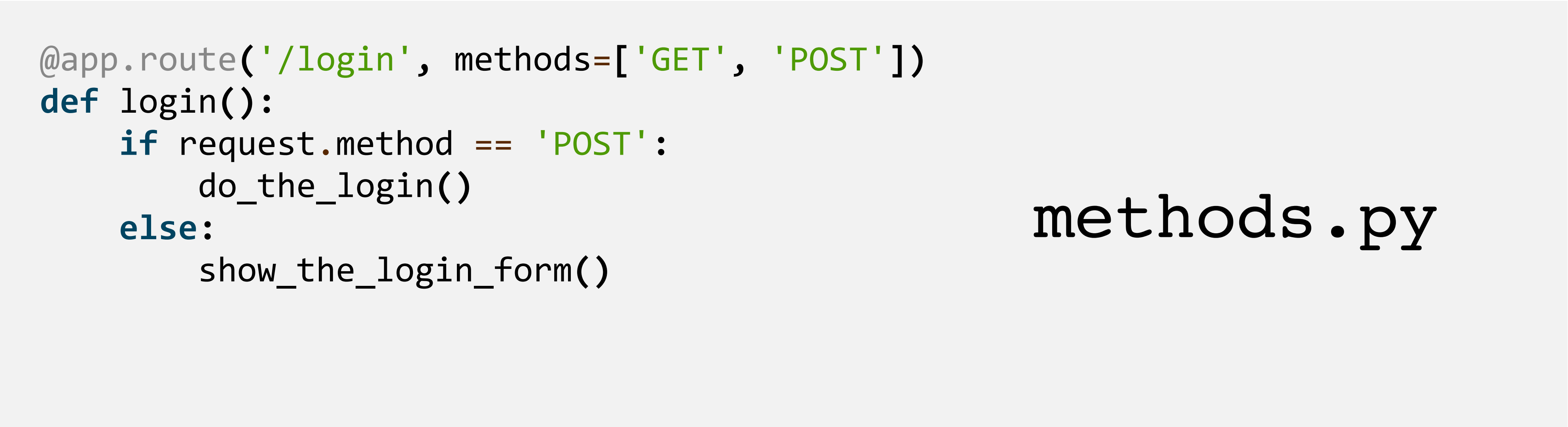
```
@app.route('/')
....
@app.route('/hello')
...
@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % username
```



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

urls.py

HTTP Methods

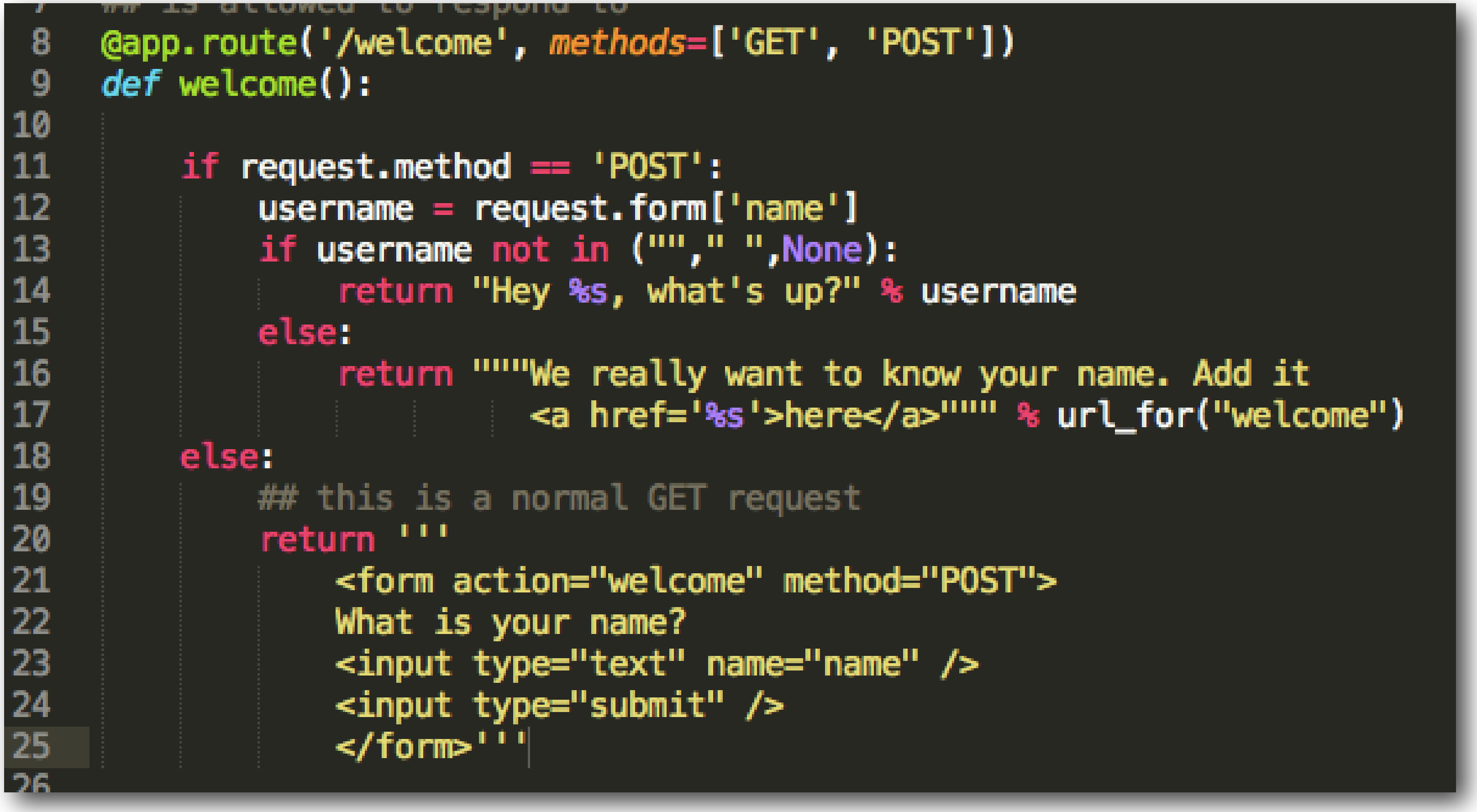


```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        do_the_login()
    else:
        show_the_login_form()
```

methods.py

Forms

form1.py



```python
## is allowed to respond to
@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>'''

```

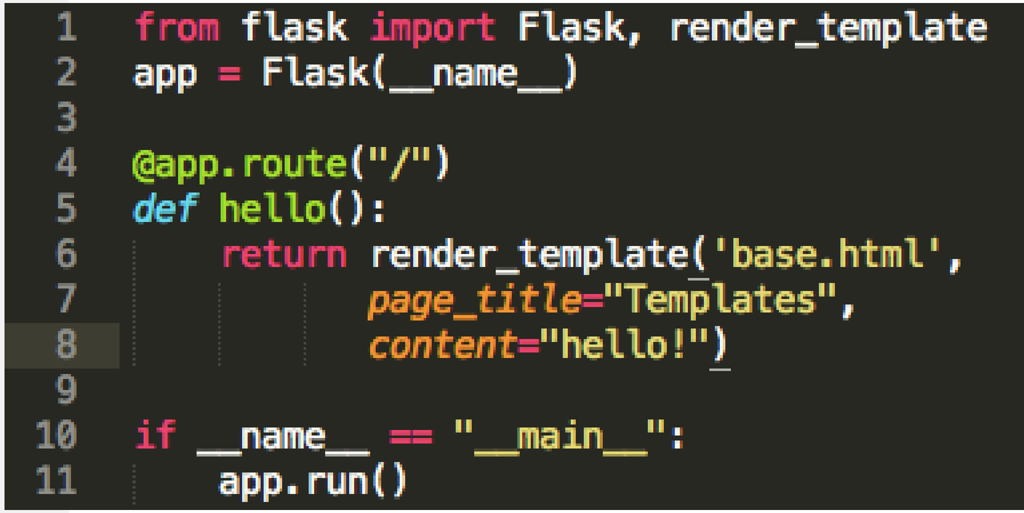
but it’s annoying to have to put HTML into Python…

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



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

@app.route("/")
def hello():
    return render_template('base.html',
           page_title="Templates",
           content="hello!")

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

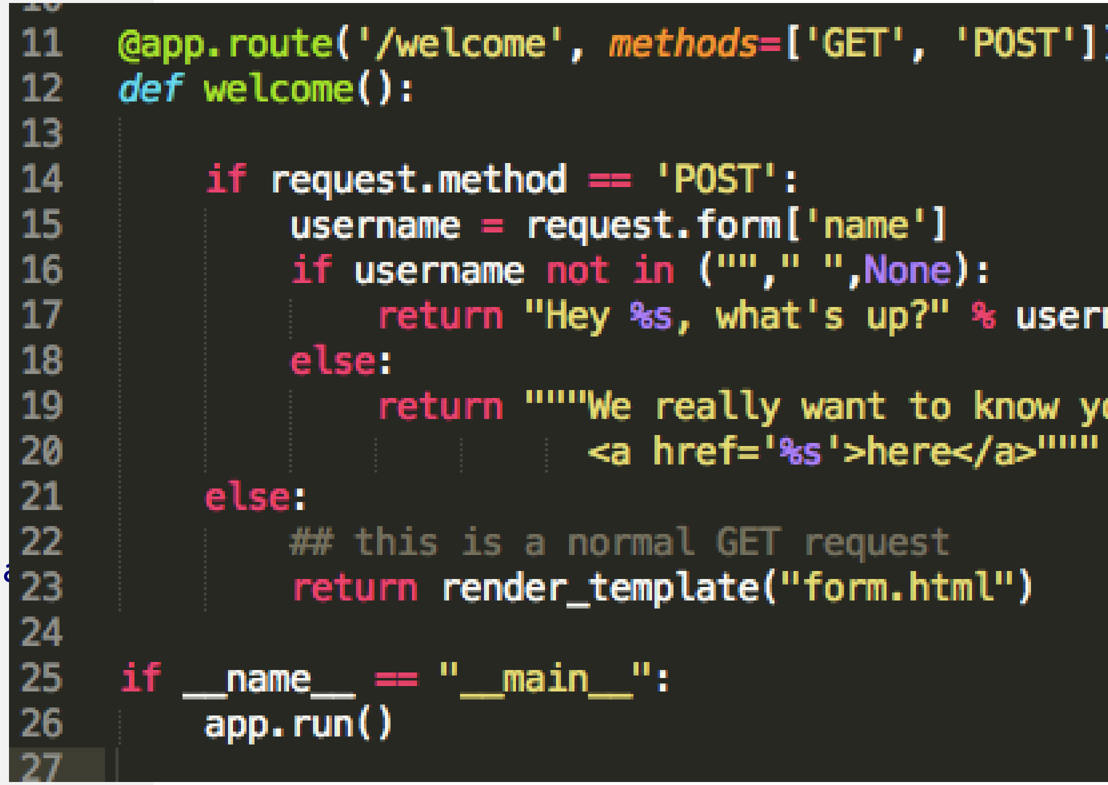
thello.py

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="na
/>
            <input type="submit" />
            </form>
{% endblock %}
```

templates/form.html



```python
@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?" % usern
        else:
            return """We really want to know yo
                    <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

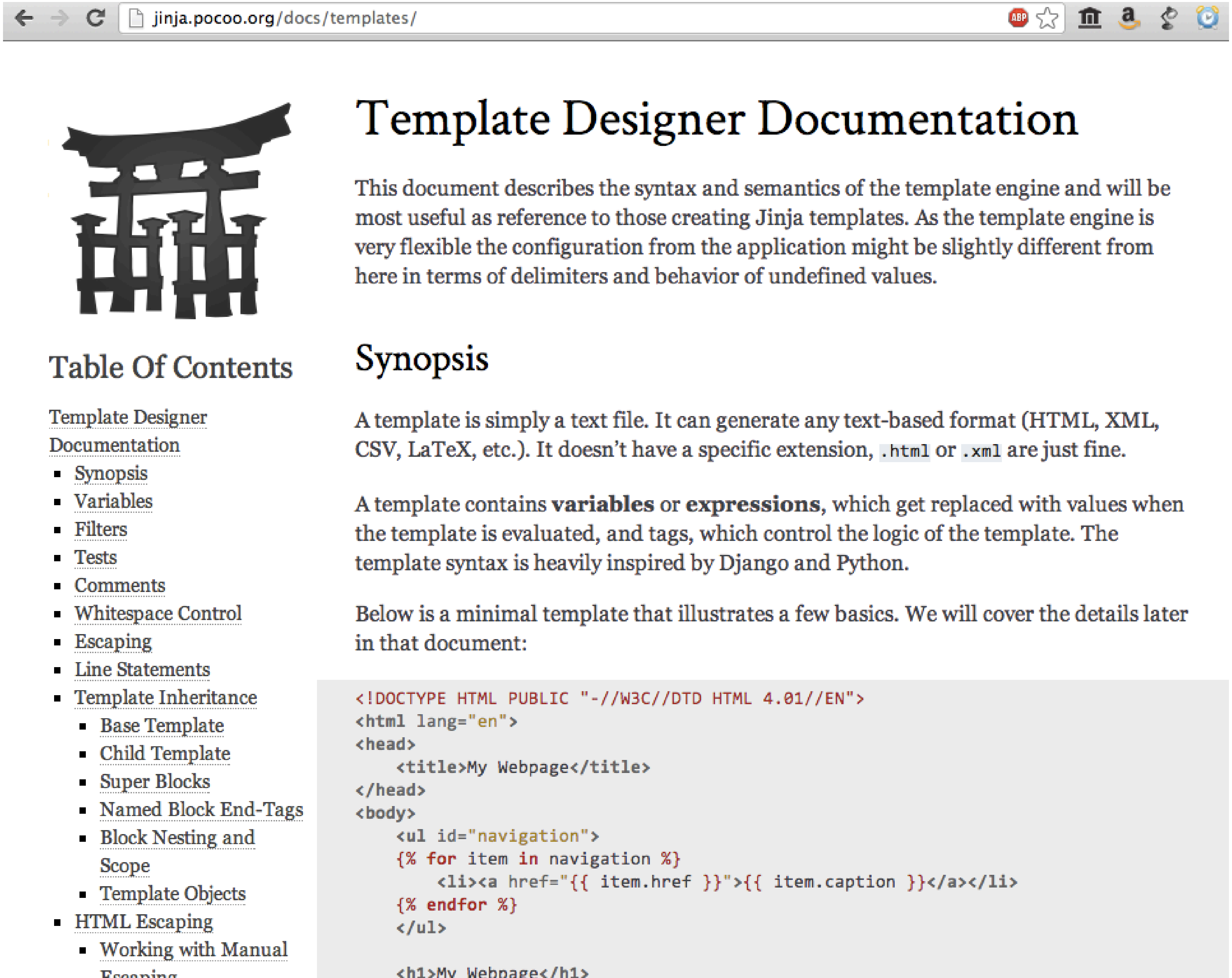


Table Of Contents

Template Designer Documentation

- Synopsis
- Variables
- Filters
- Tests
- Comments
- Whitespace Control
- Escaping
- Line Statements
- Template Inheritance
  - Base Template
  - Child Template
  - Super Blocks
  - Named Block End-Tags
  - Block Nesting and Scope
  - Template Objects
- HTML Escaping
  - Working with Manual Escaping

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

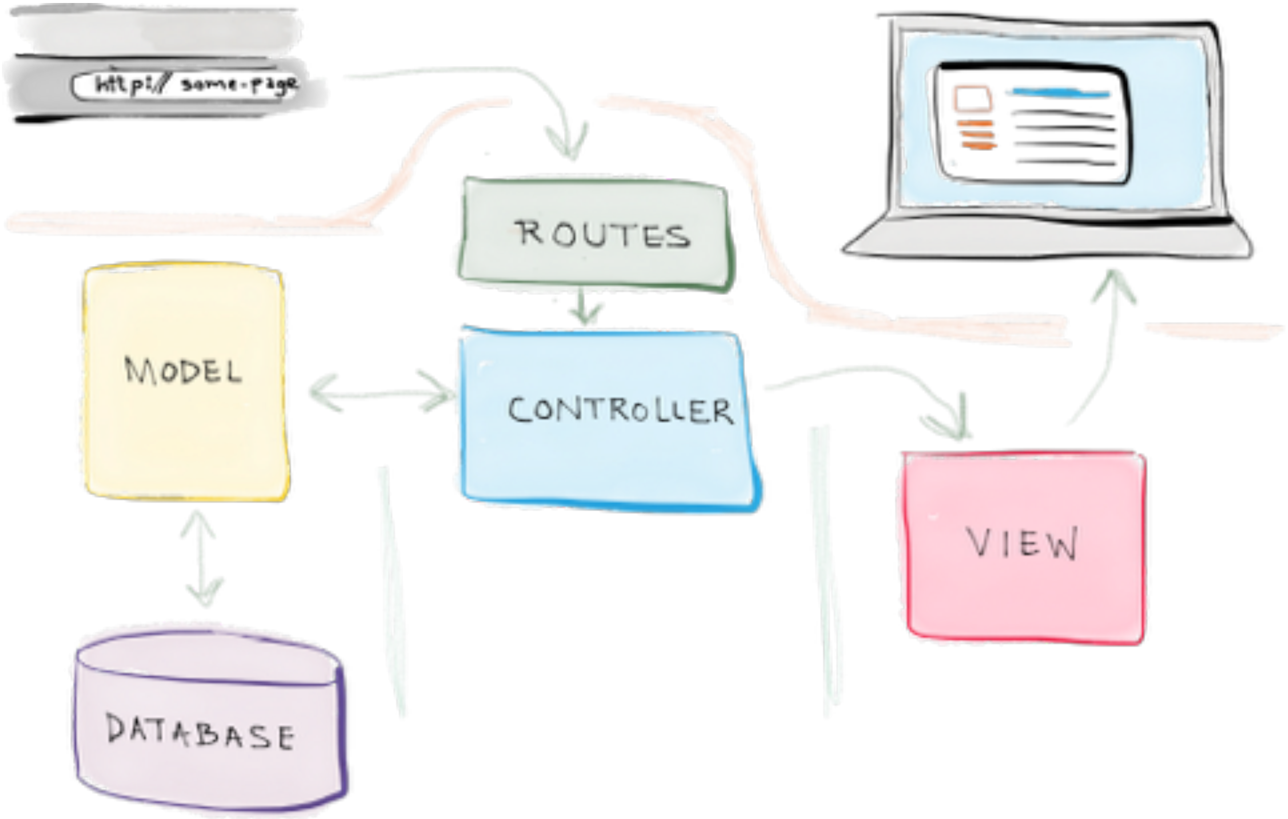
A template is simply a text file. It can generate any text-based format (HTML, XML, CSV, LaTeX, etc.). It doesn't have a specific extension, `.html` or `.xml` are just fine.

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...



https://realpython.com/blog/python/the-model-view-controller-mvc-paradigm-summarized-with-legos/

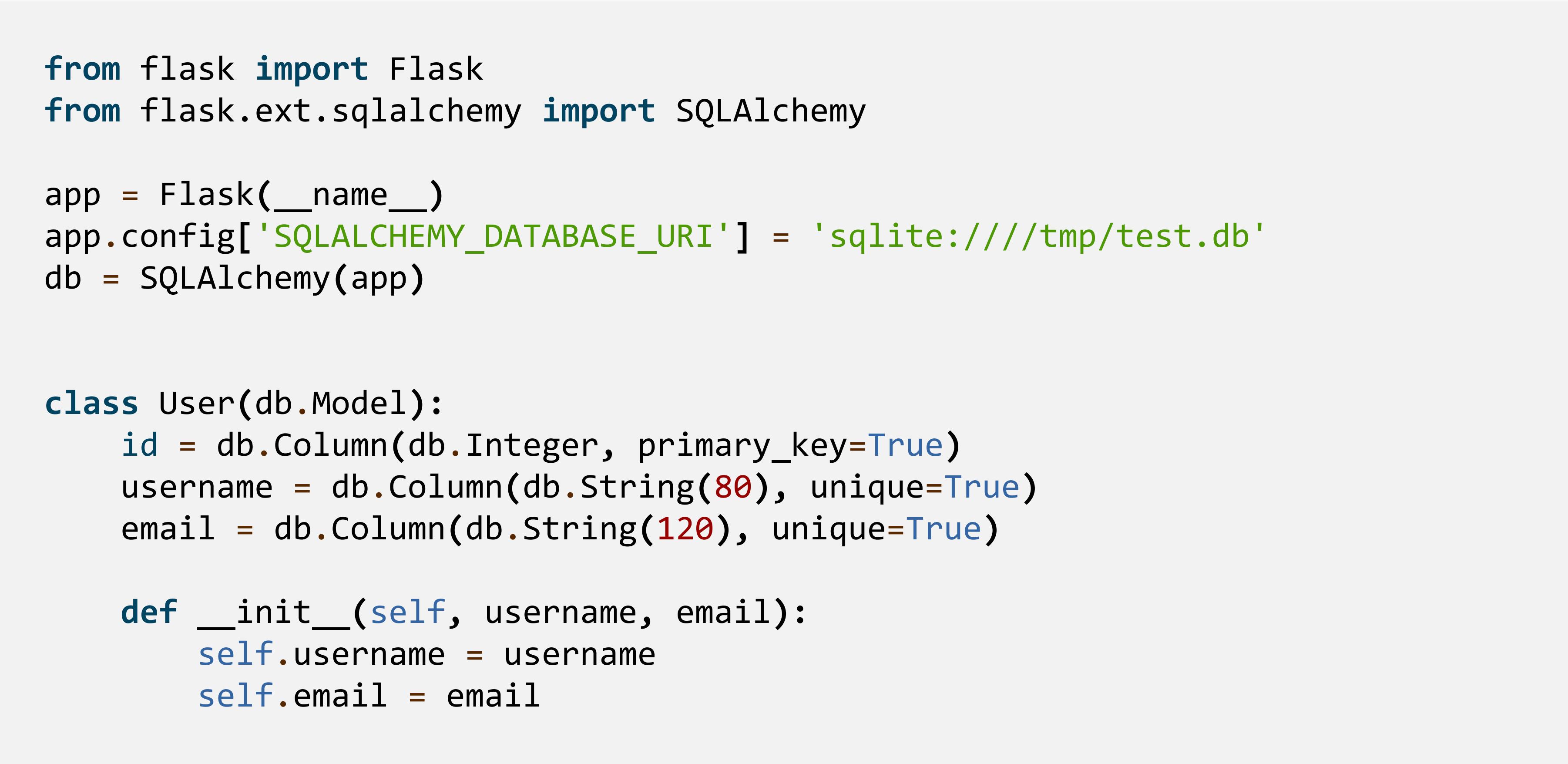
Flask can behave something like it with SQLAlchemy...

```
pip install flask-sqlalchemy
```

Models: This is where data goes.

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

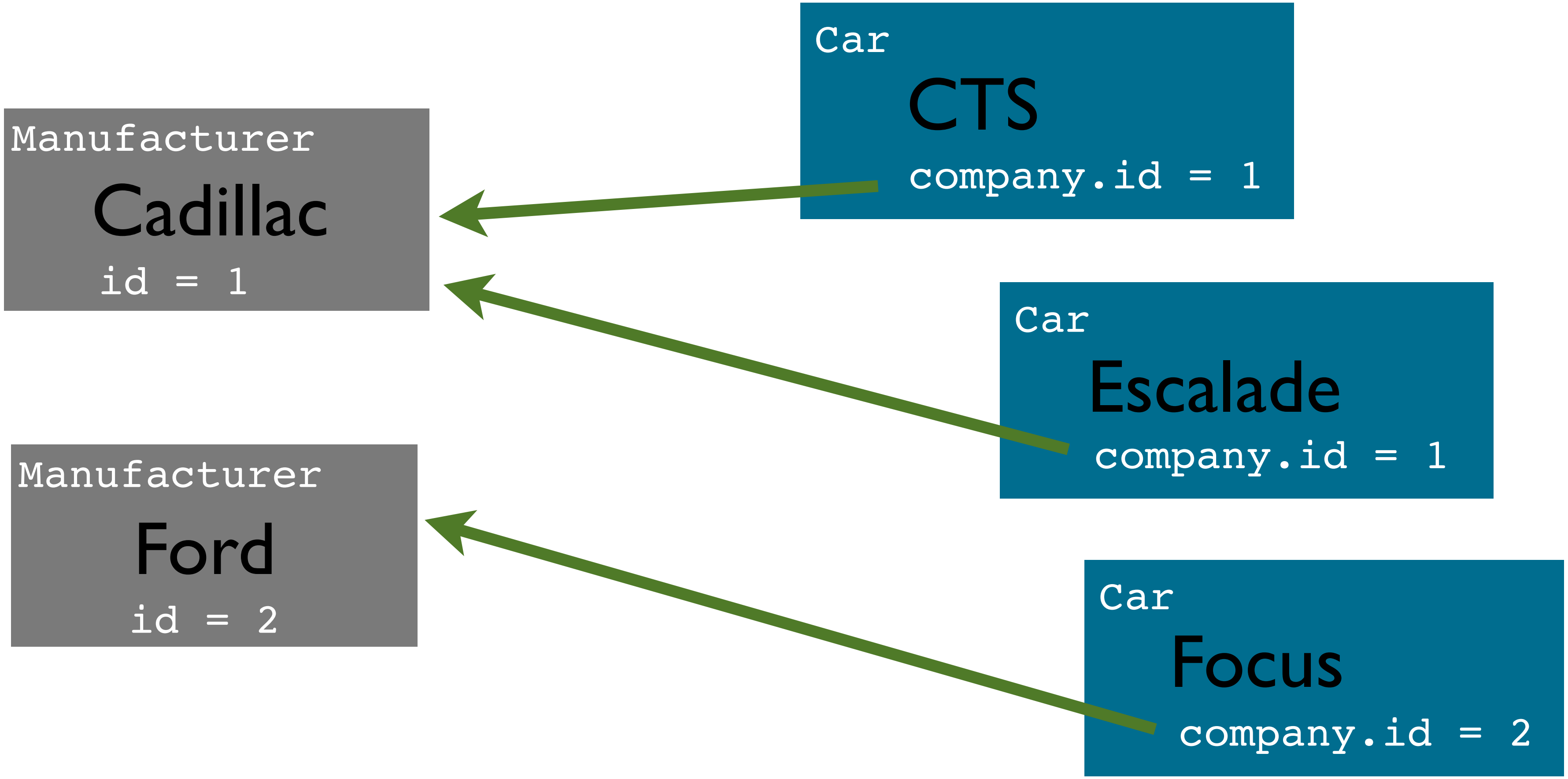
Stored in whatever database your config sets.



```
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')
```



Manufacturer

Cadillac

id = 1

Manufacturer

Ford

id = 2

Car

CTS

company.id = 1

Car

Escalade

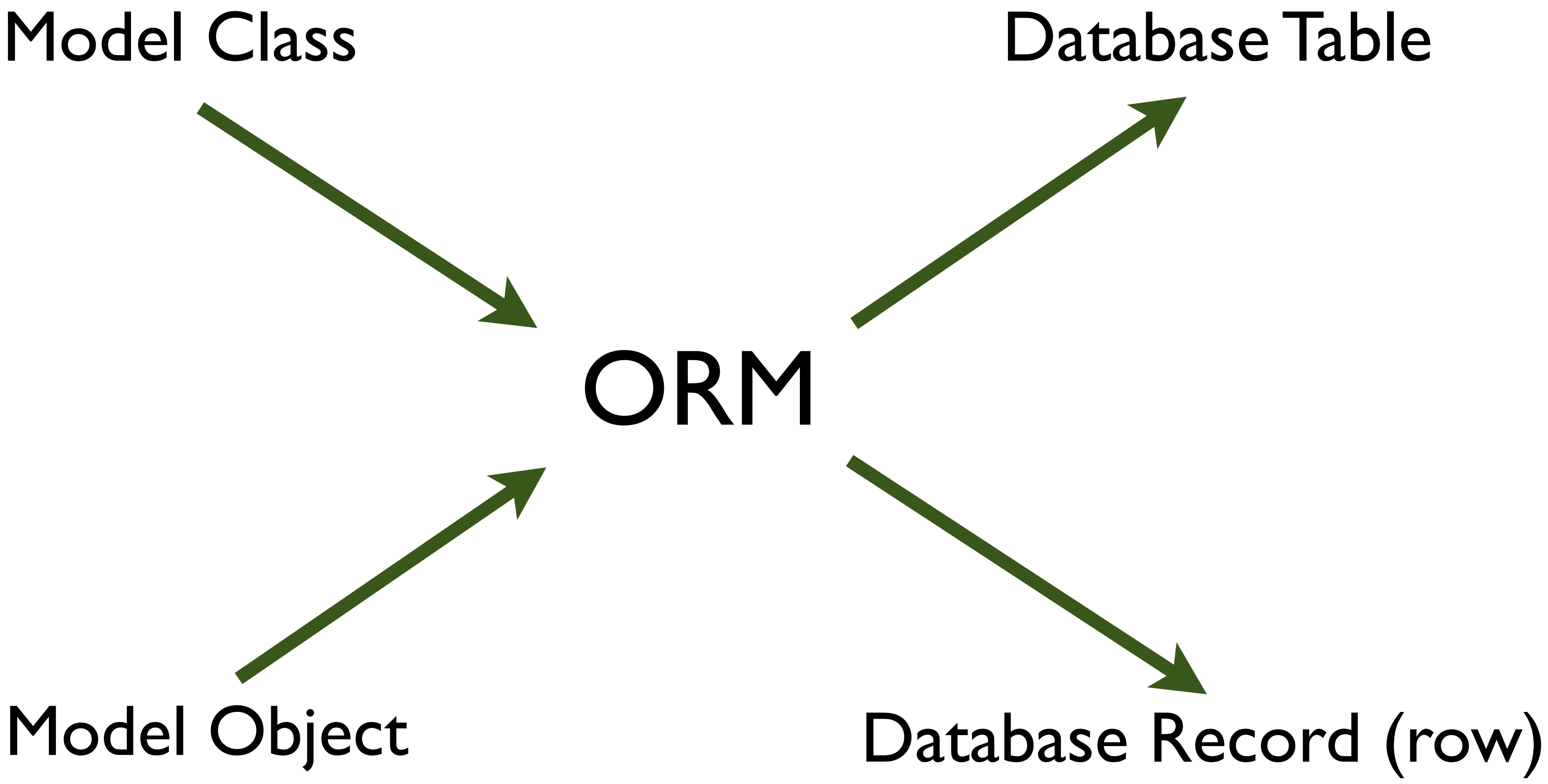
company.id = 1

Car

Focus

company.id = 2

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)

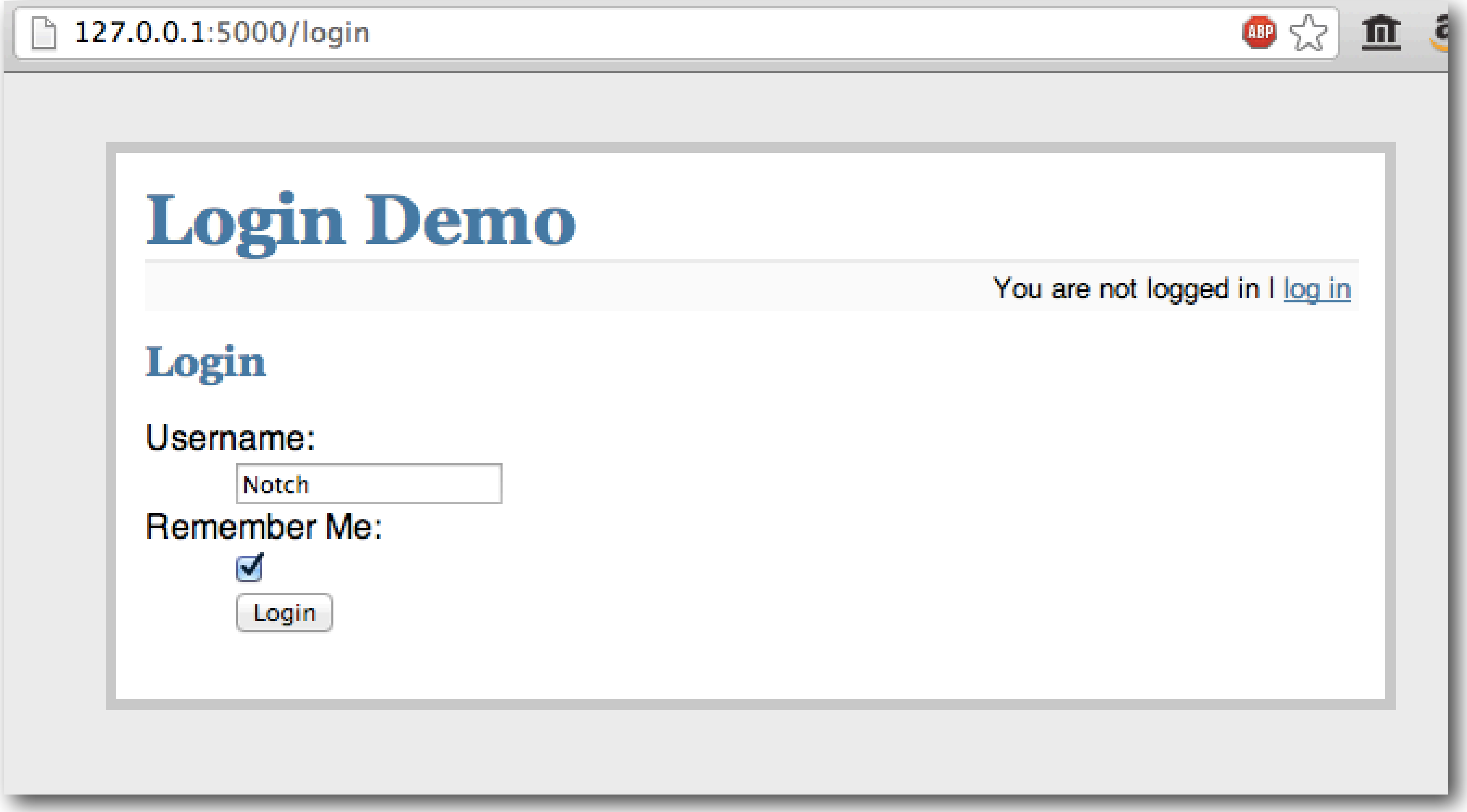
OpenID

Sending Mail (Flask-Mail)

Themes

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

```
pip install flask-login
```



127.0.0.1:5000/login

Login Demo

You are not logged in | log in

Login

Username:

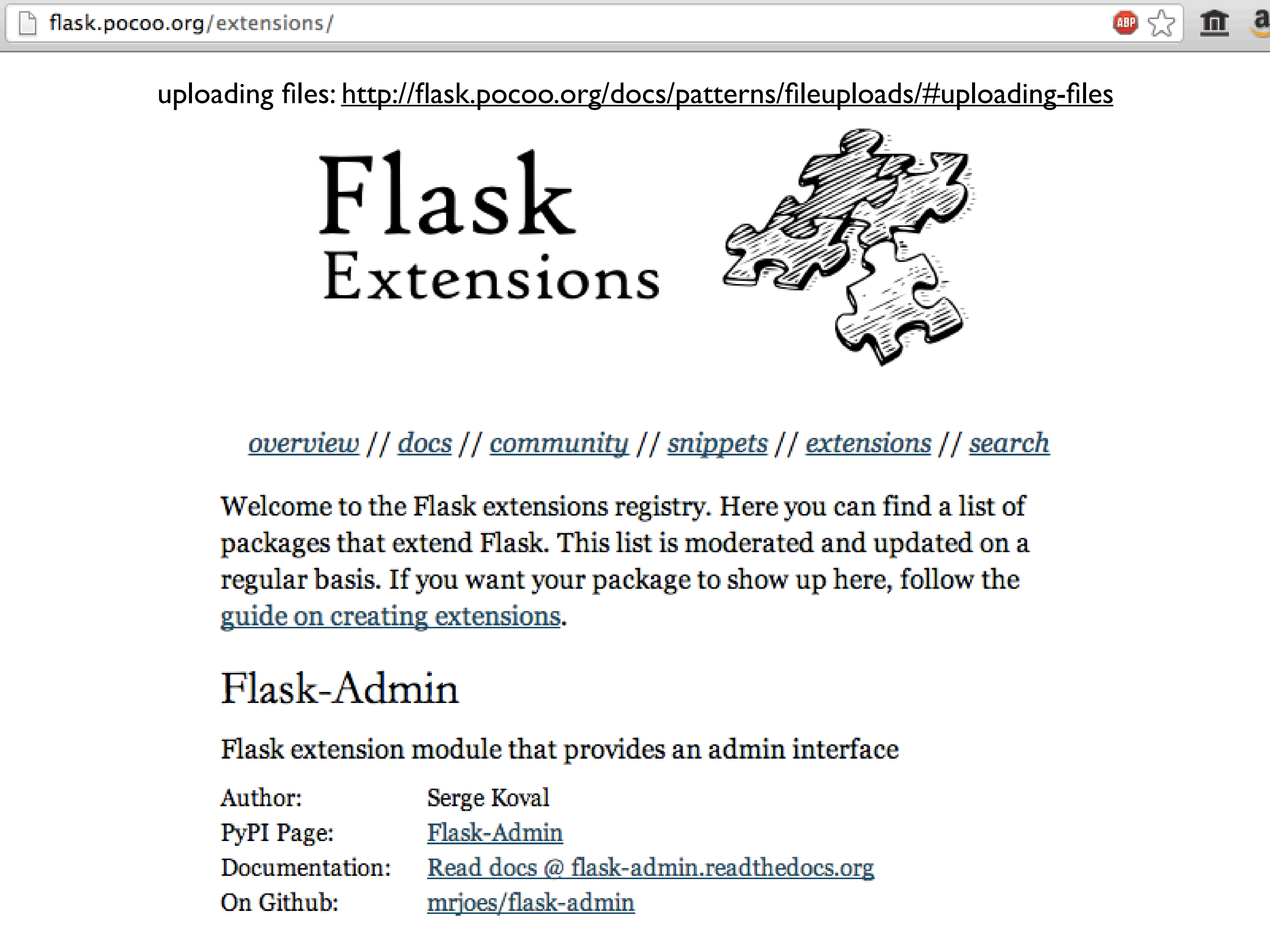
Notch

Remember Me:

☑

Login

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



flask.pocoo.org/extensions/

Flask Extensions

overview // docs // community // snippets // extensions // search

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.

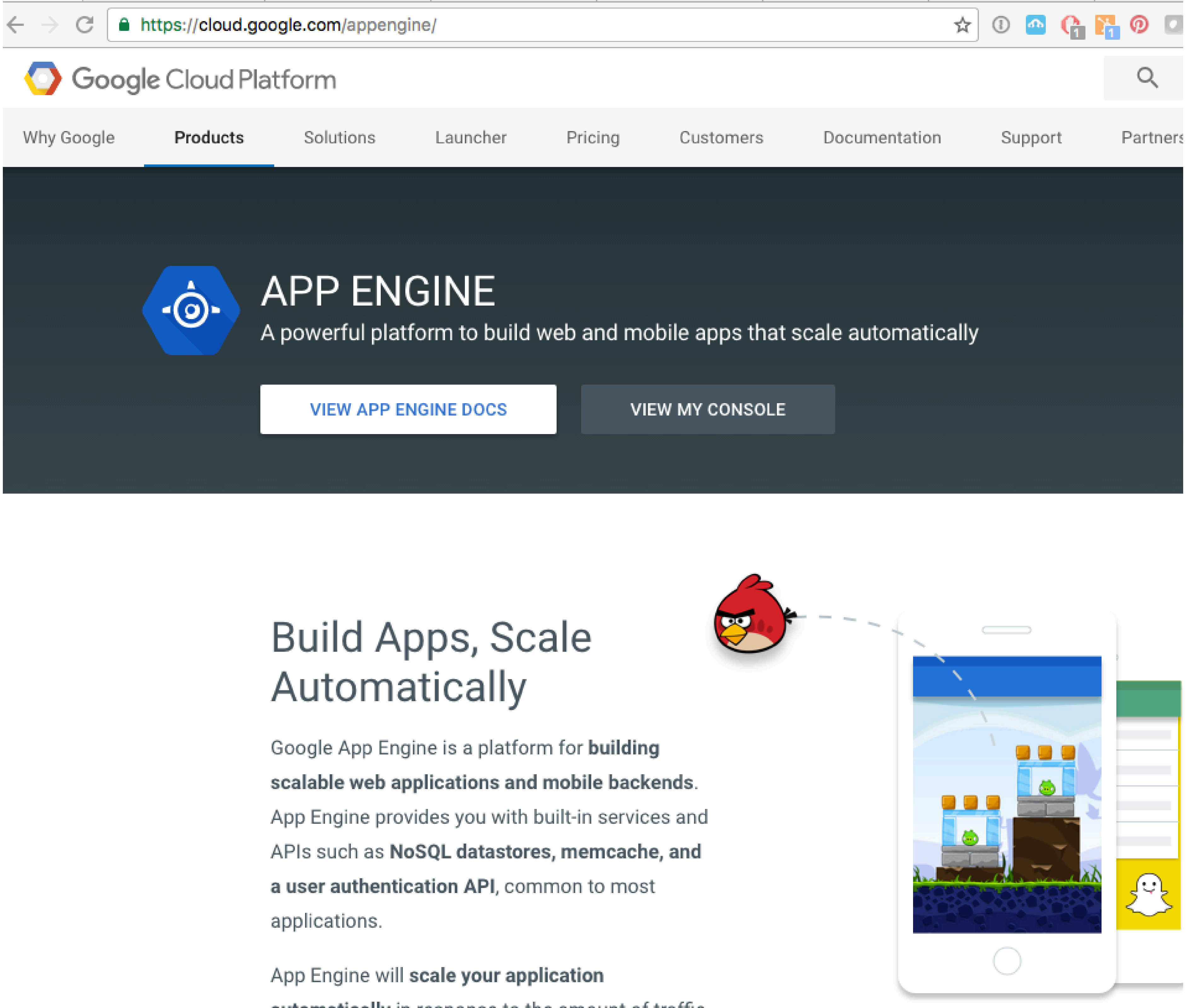
Flask-Admin

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?
journal.py



APP ENGINE

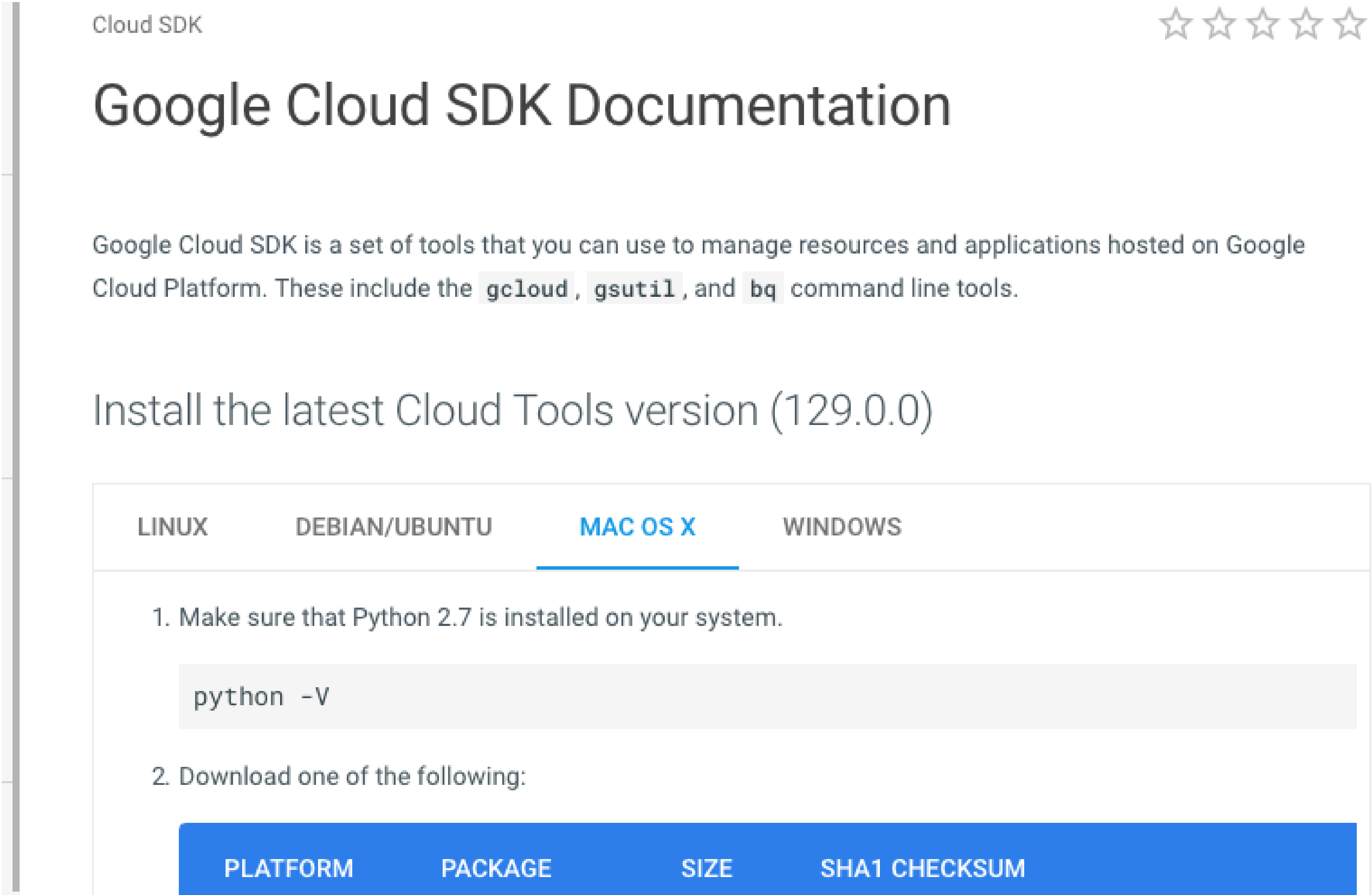
A powerful platform to build web and mobile apps that scale automatically

Build Apps, Scale Automatically

Google App Engine is a platform for **building scalable web applications and mobile backends**. App Engine provides you with built-in services and APIs such as **NoSQL datastores, memcache, and a user authentication API**, common to most applications.

App Engine will **scale your application automatically** in response to the amount of traffic

SDK allows you to run a local version of your projects



Cloud SDK

☆☆☆☆☆

Google Cloud SDK Documentation

Google Cloud SDK is a set of tools that you can use to manage resources and applications hosted on Google Cloud Platform. These include the `gcloud`, `gsutil`, and `bq` command line tools.

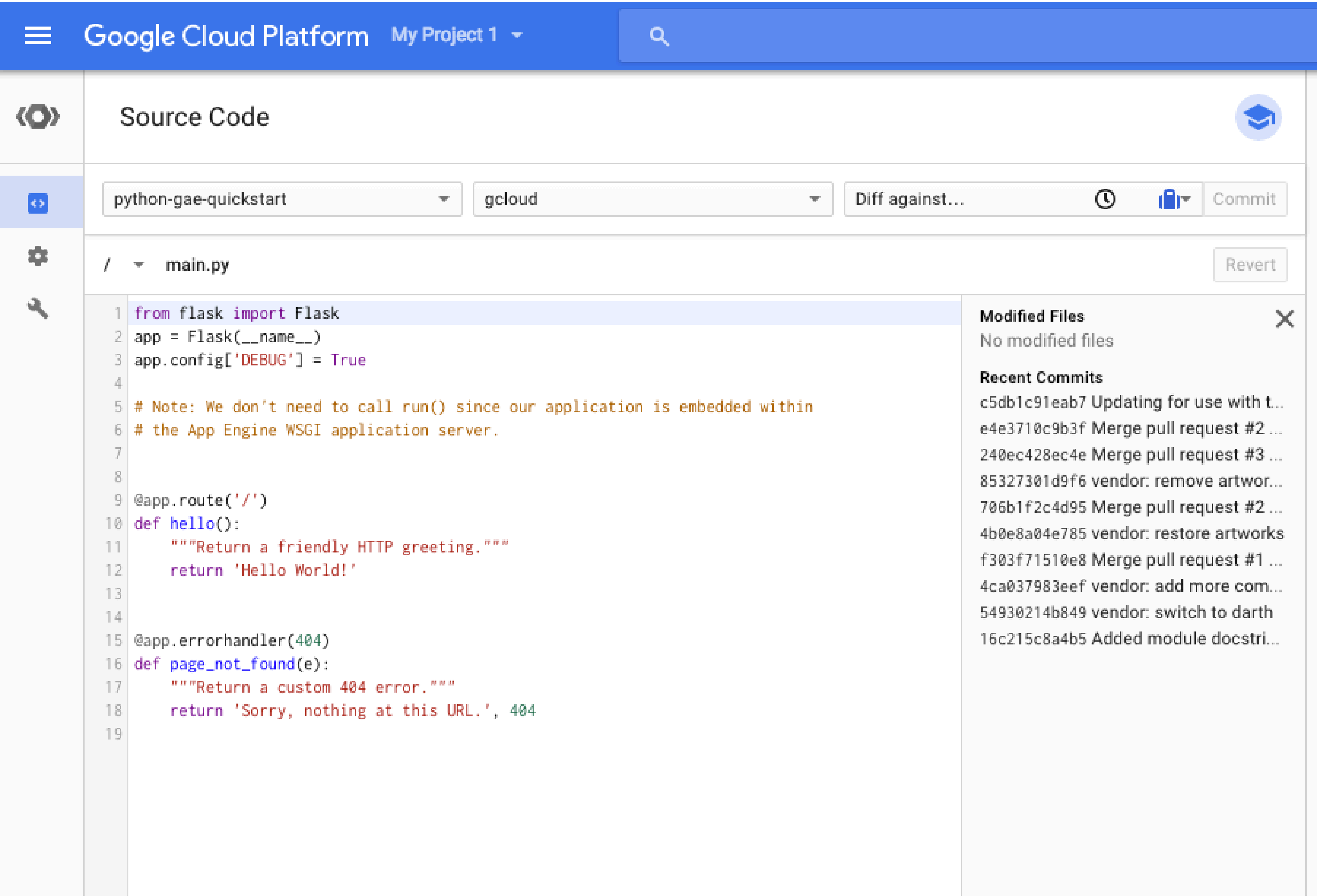
Install the latest Cloud Tools version (129.0.0)

1. Make sure that Python 2.7 is installed on your system.

```
python -V
```

2. Download one of the following:

PLATFORM	PACKAGE	SIZE	SHA1 CHECKSUM



python-gae-quickstart

gcloud

Diff against…

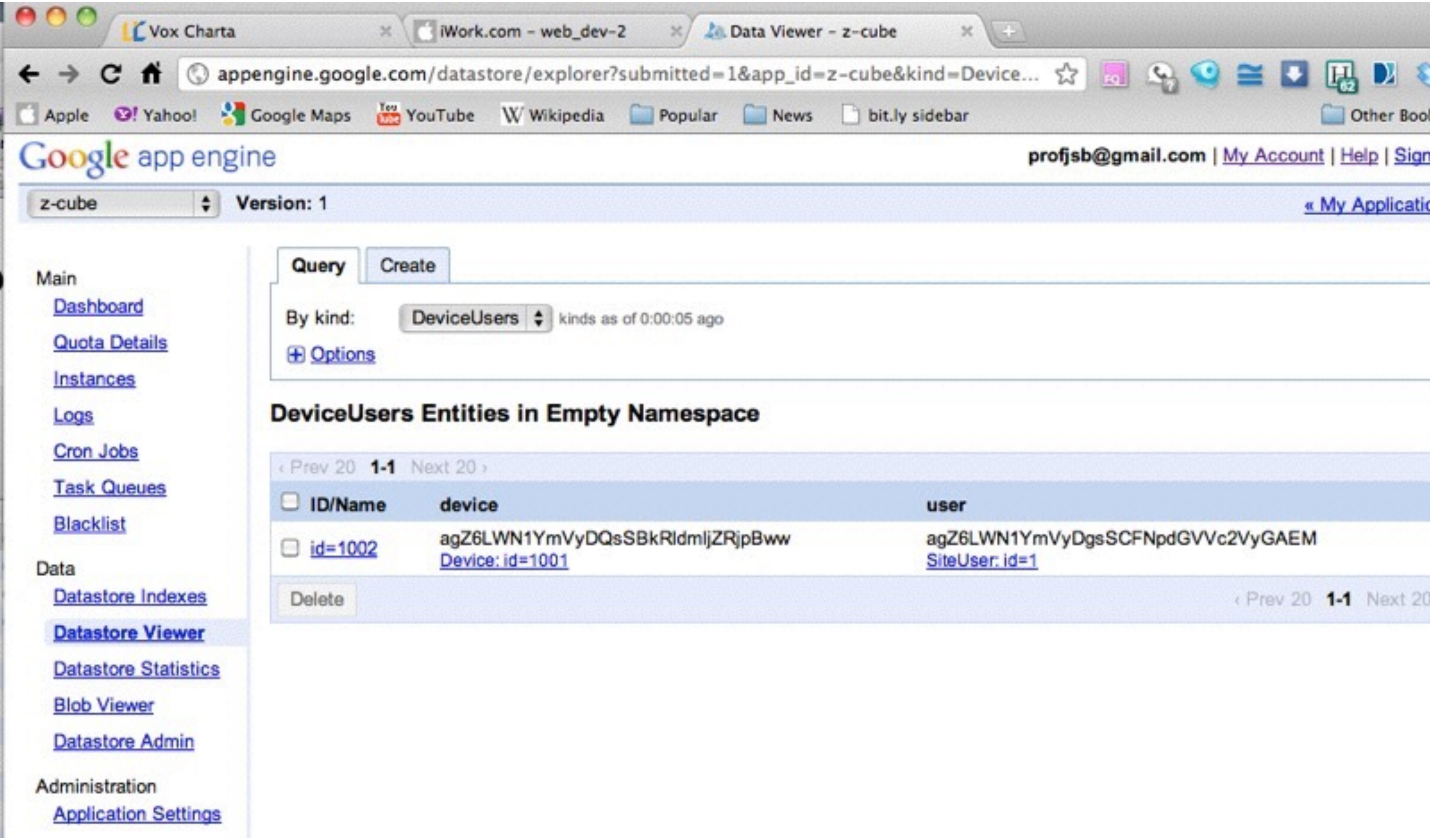
main.py

```
from flask import Flask
app = Flask(__name__)
app.config['DEBUG'] = True

# Note: We don't need to call run() since our application is embedded within
# the App Engine WSGI application server.


@app.route('/')
def hello():
    """Return a friendly HTTP greeting."""
    return 'Hello World!'


@app.errorhandler(404)
def page_not_found(e):
    """Return a custom 404 error."""
    return 'Sorry, nothing at this URL.', 404

```

No modified files

```
c5db1c91eab7 Updating for use with t...
e4e3710c9b3f Merge pull request #2 ...
240ec428ec4e Merge pull request #3 ...
85327301d9f6 vendor: remove artwor...
706b1f2c4d95 Merge pull request #2 ...
4b0e8a04e785 vendor: restore artworks
f303f71510e8 Merge pull request #1 ...
4ca037983eef vendor: add more com...
54930214b849 vendor: switch to darth
16c215c8a4b5 Added module docstri...
```

online management...



z-cube Version: 1

By kind: DeviceUsers kinds as of 0:00:05 ago

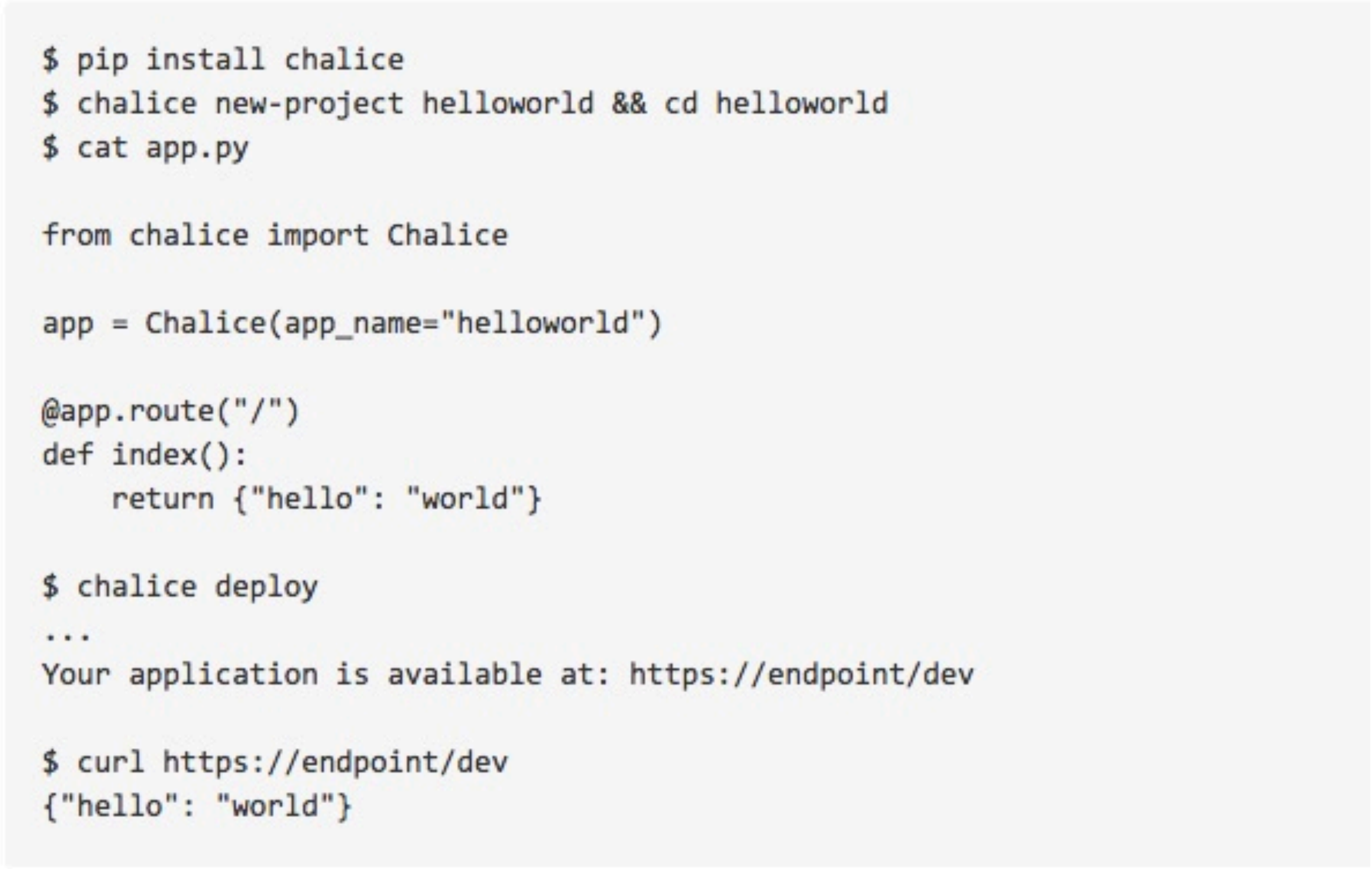
DeviceUsers Entities in Empty Namespace

‹ Prev 20 **1-1** Next 20 ›

□ ID/Name	device	user
□ id=1002	agZ6LWN1YmVyDQsSBkRldmljZRjpBww Device: id=1001	agZ6LWN1YmVyDgsSCFNpdGVVc2VyGAEM SiteUser: id=1

‹ Prev 20 **1-1** Next 20

Python Serverless Microframework for AWS



```
$ pip install chalice
$ chalice new-project helloworld && cd helloworld
$ cat app.py

from chalice import Chalice

app = Chalice(app_name="helloworld")

@app.route("/")
def index():
    return {"hello": "world"}

$ chalice deploy
...
Your application is available at: https://endpoint/dev

$ curl https://endpoint/dev
{"hello": "world"}
```

https://github.com/awslabs/chalice