

Computer Networks The Heralds Of Resource Sharing 1972 Arpanet Do

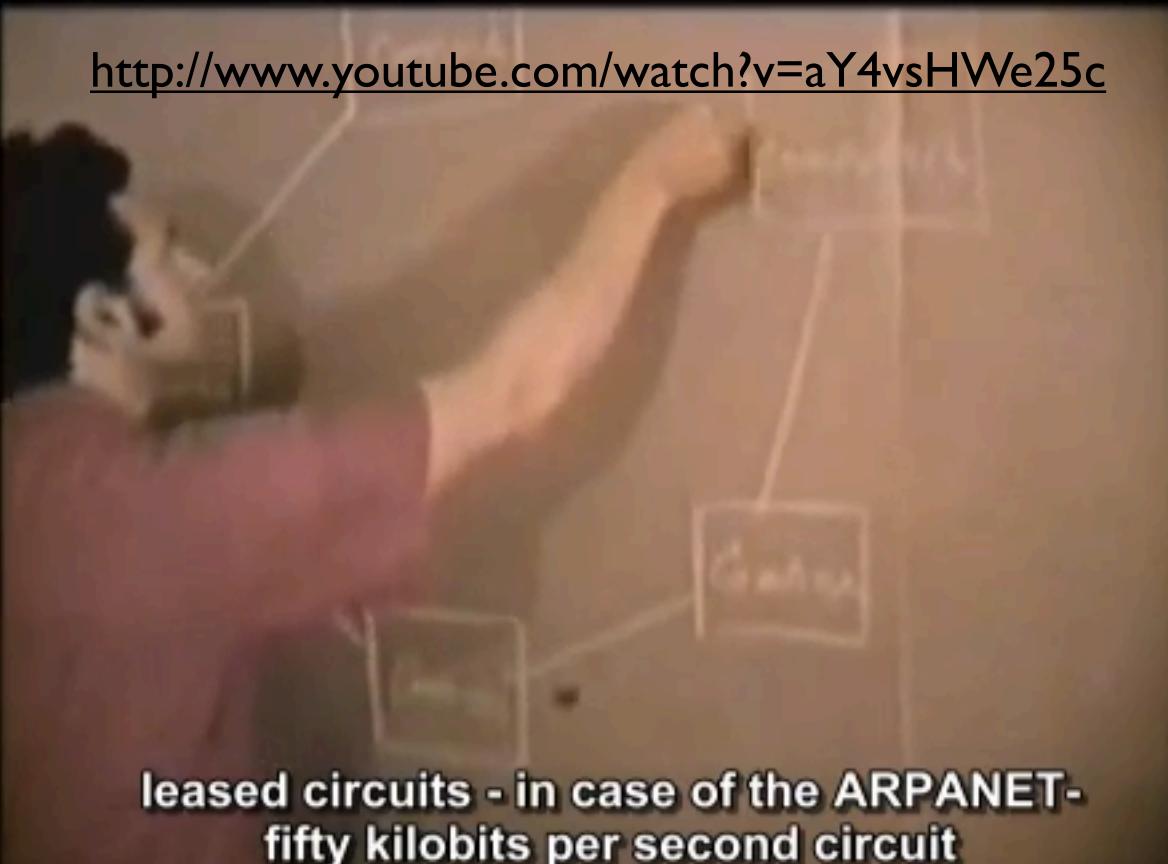
Dat Pham



Subscribe

1,107 videos ▾

<http://www.youtube.com/watch?v=aY4vsHWe25c>



leased circuits - in case of the ARPANET-
fifty kilobits per second circuit



03:51 / 25:36



Like



Add to

Share



563 views



Uploaded by Dat Pham on Oct 6, 2011

No description available.

10 likes, 0 dislikes

Web Development with Python

AY250 Fall 2012



<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

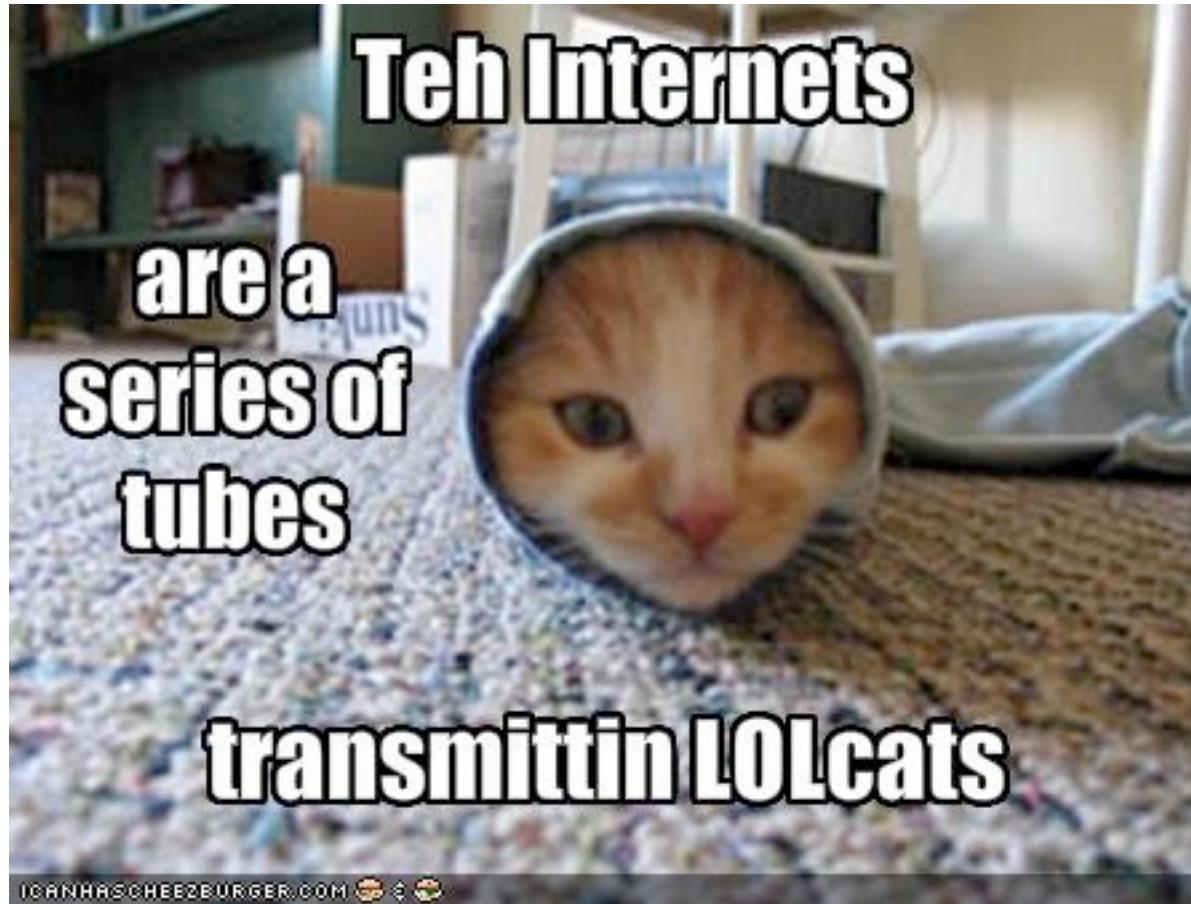
Using Python for web supremacy

Basic Python servers

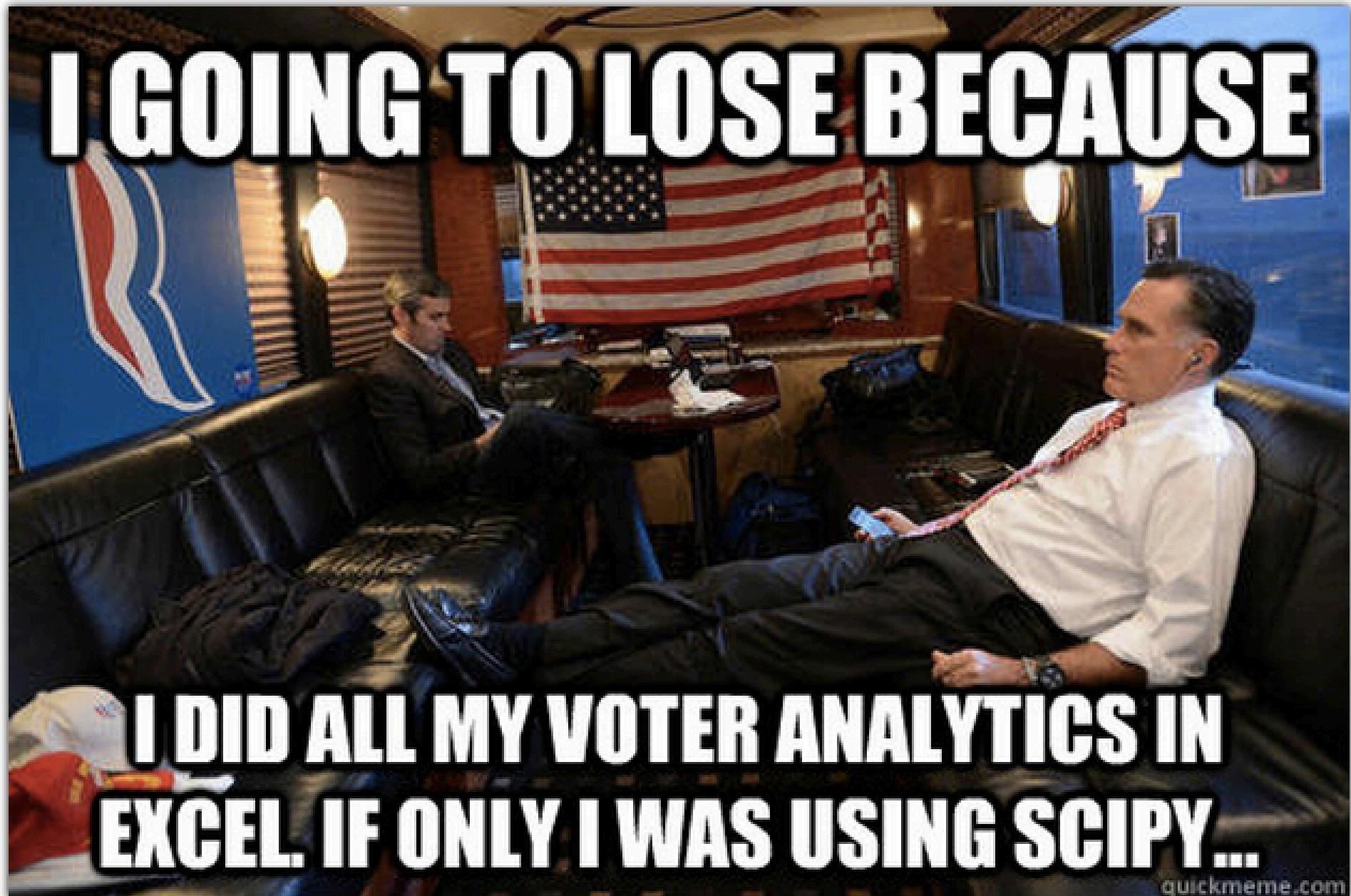
Frameworks and using Flask

Google App Engine

Believe it or not



the internet was not built for this.



quickmeme.com

<http://www.quickmeme.com/meme/3rmnpe/>

Sudden Realization Romney

I GOING TO LOSE BECAUSE



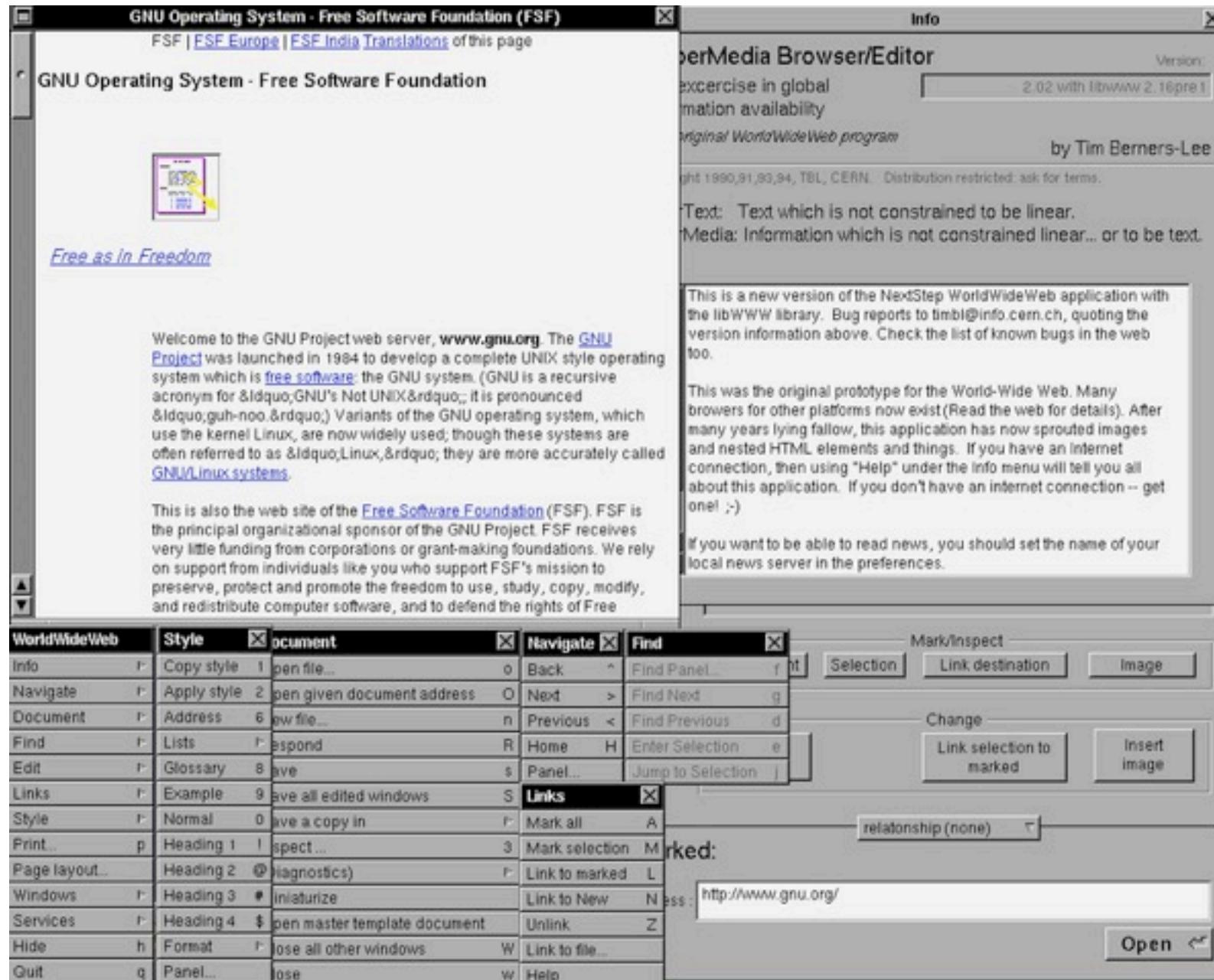
quickmeme.com

<http://www.quickmeme.com/meme/3rmnol/>

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



The browser model



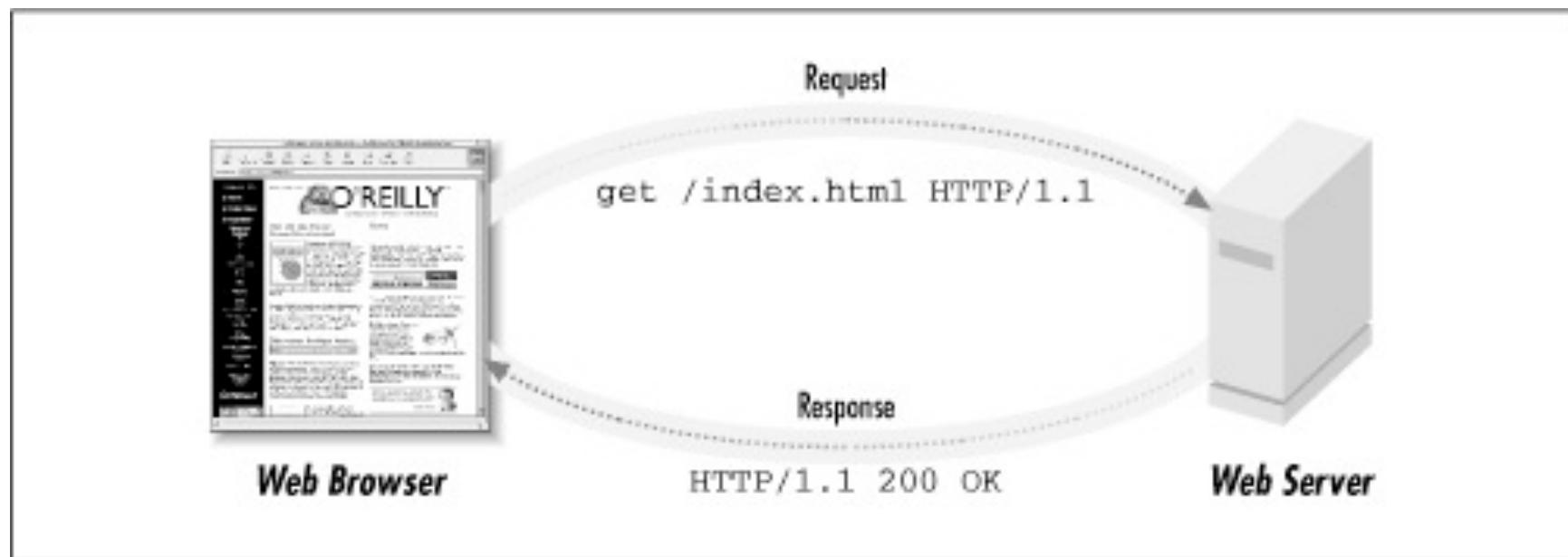
Request and Response

User knocks on the server's door
(request)

Server thinks about it

Server sends data back to the user
(response)

Request and Response



Browser stuff

Google code Search
e.g. "ajax apis" or "open source"

★ Google Code University

[Home](#)
[Tutorials and Introductions](#)
[Courses](#)
[Programming Languages](#)
[Web Programming](#)
[Web Security](#)
[Algorithms](#)
[Android](#)
[Distributed Systems](#)
[Tools 101](#)
[Google APIs & Tools](#)
[Discussion Forums](#)
[Submit a Course](#)
[Curriculum Search New!](#)

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

Google HTML/CSS/Javascript from the Ground Up.
Why?

- Accessibility
- Portability 
- Maintainability
- Reduced latency

▶ ⏪ 0:00 / 6:12 ⏹ ⏷

<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

Simple Servers

Recall the XML/RPC Server...

a webserver just responds to a different request protocol

```
import BaseHTTPServer
```

Simple Servers

Recall the XML/RPC Server...

a webserver just responds to a different request protocol

```
import BaseHTTPServer  
class myresponse(BaseHTTPServer.BaseHTTPRequestHandler):
```

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):
```

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>" )
```

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)
```

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()
```

Simple Servers

Recall the XML/RPC Server...

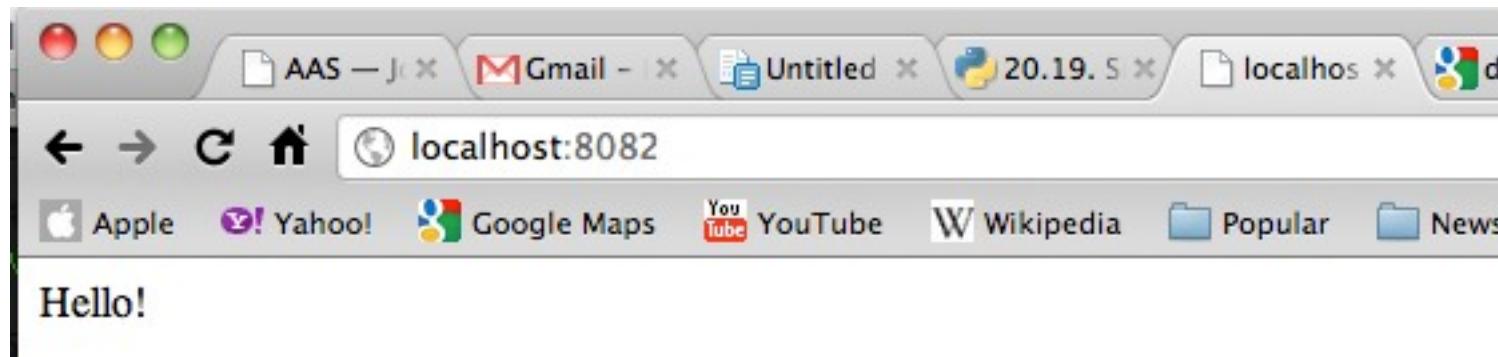
a webserver just responds to a different request protocol

```
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

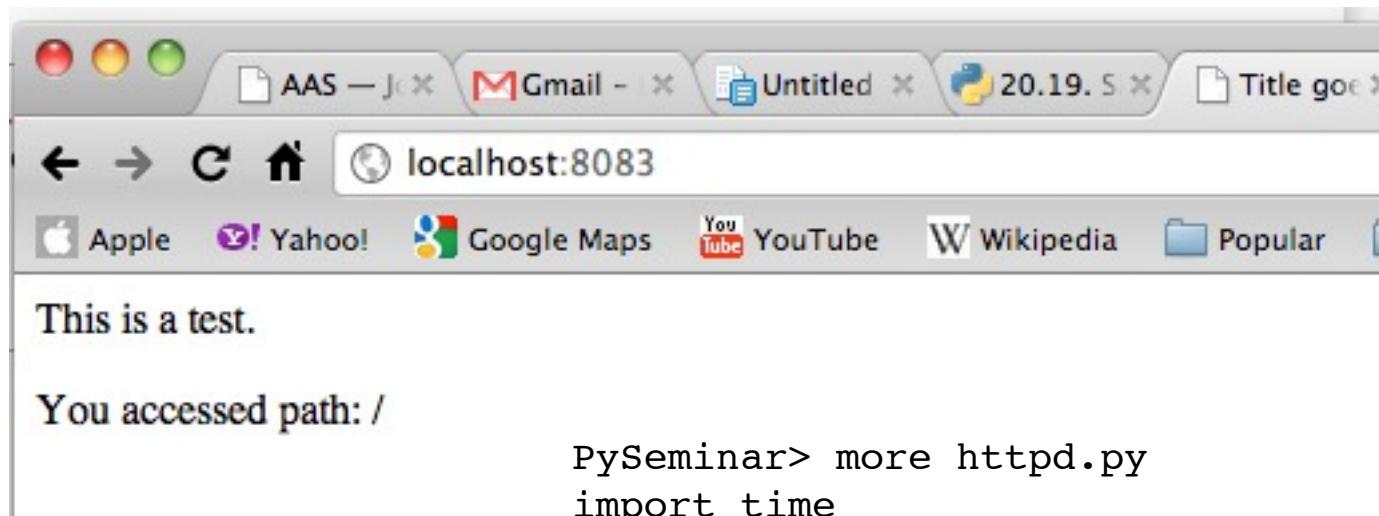
note: BaseHTTPServer is not threaded



```
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
```

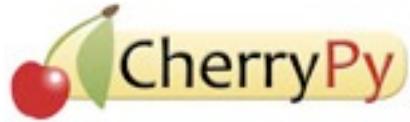
```
PySeminar> more httpd.py
import time
import BaseHTTPServer
HOST_NAME = 'localhost' # !!!REMEMBER TO CHANGE THIS!!!
PORT_NUMBER = 8083 # Maybe set this to 9000.
class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_HEAD(s):
        s.send_response(200)
        s.send_header("Content-type",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

```
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



: easy_install cherrypy

Simple Servers



: easy_install cherrypy

```
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></i>'
greetUser.exposed = True

cherrypy.quickstart\(WelcomePage\(\)\)
```

file: cp1.py

Simple Servers



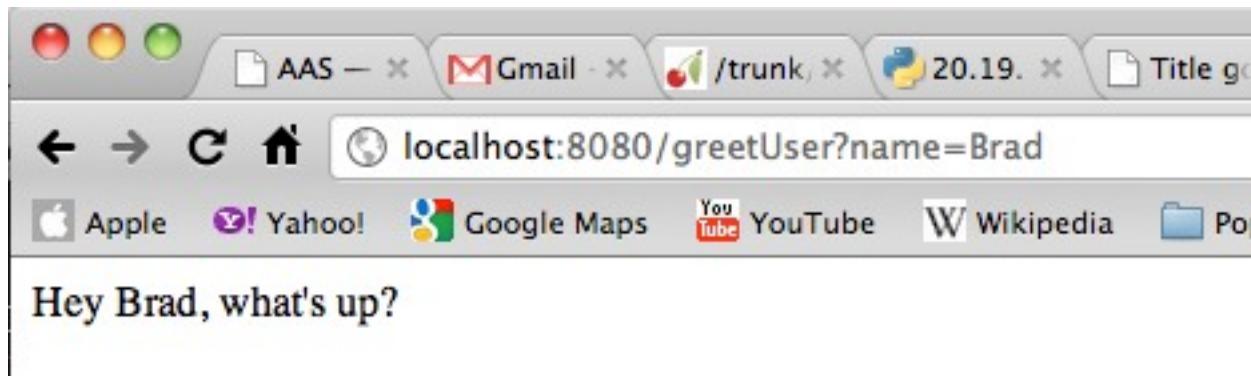
: easy_install cherrypy

```
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></i>'
greetUser.exposed = True

cherrypy.quickstart\(WelcomePage\(\)\)
```

file: cp1.py

here, name is a variable of the GET request



Simple Servers



>>> run cp2

A screenshot of a Mac OS X desktop. The browser window title is "localhost:8080". The page content is a simple form:

```
What is your name?  
```

```
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
```

Simple Servers



>>> run cp2

A screenshot of a Mac OS X desktop. In the top menu bar, there are icons for AAS, Gmail, /trunk, and 20.19. In the center, the address bar shows 'localhost:8080'. Below the address bar, there are links for Apple, Yahoo!, Google Maps, YouTube, and Wikipedia. The main content area of the browser shows a form with the question 'What is your name?' and a text input field containing 'Jerry Brown'. There is also a 'Submit' button.

```
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
```

A screenshot of the same Mac OS X desktop as the previous image. The browser window now shows the URL 'localhost:8080/greetUser?name=Jerry+Brown'. The main content area of the browser displays the greeting 'Hey Jerry Brown, what's up?'

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



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

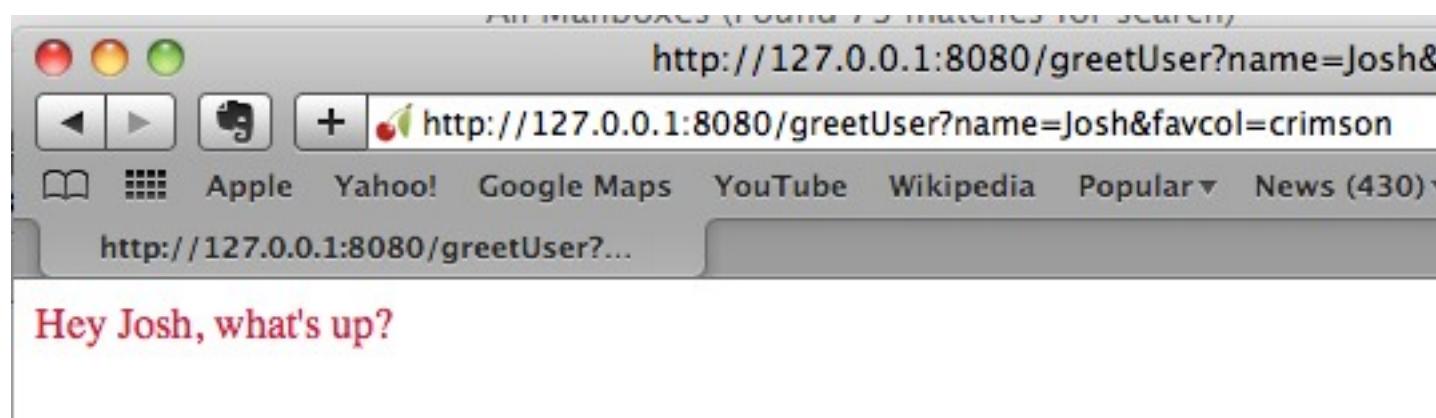
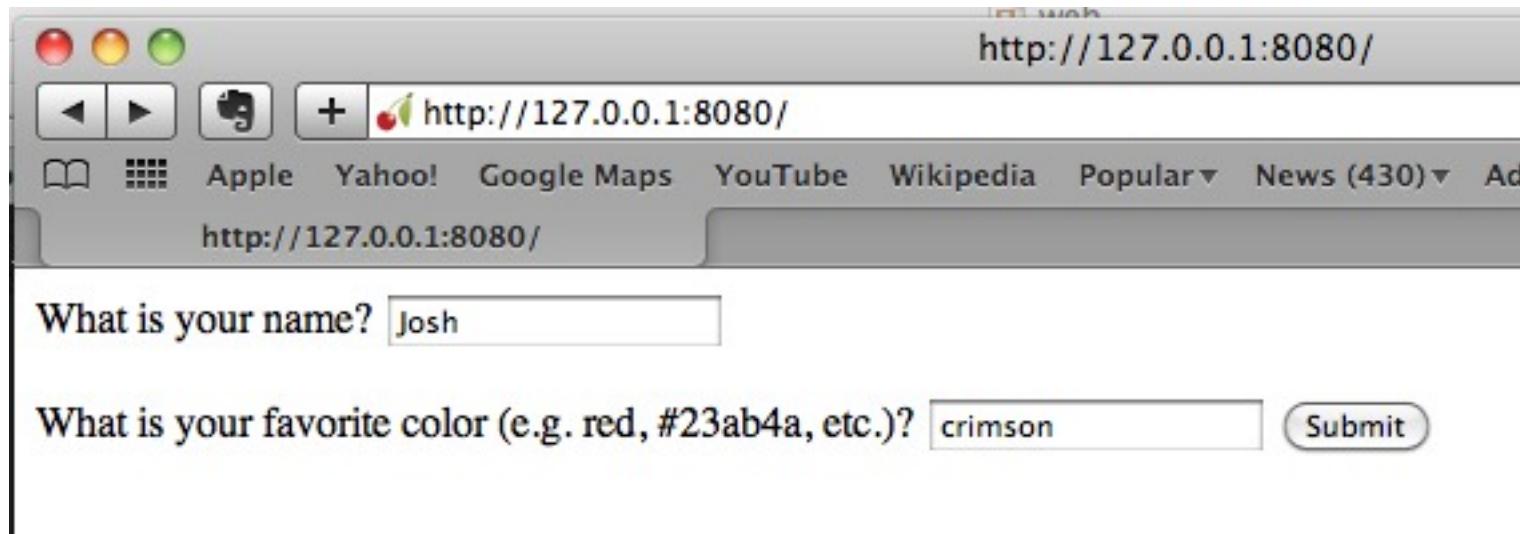


```
$ gem install localtunnel  
$ localtunnel 8000  
  
share this url:  
http://xyz.localtunnel.com
```

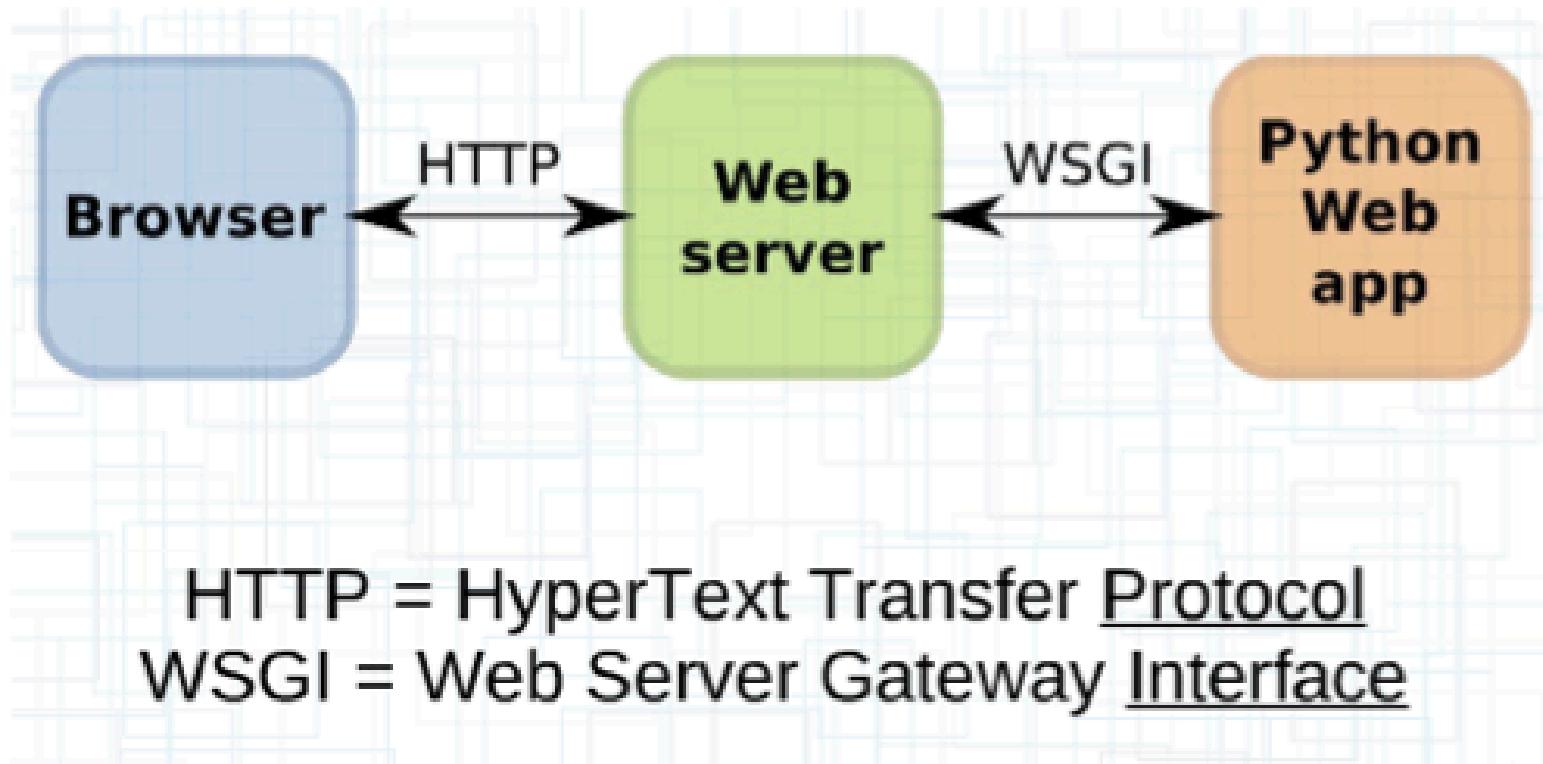
Small exercise:

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

output



What is WSGI?



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

<http://www.docstoc.com/docs/69863691/WSGI-from-Start-to-Finish>

WSGI & HTTP

```
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"),
}
```

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/>

```
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'''
```

part of the standard library

Frameworks

Don't spend time writing code for common tasks.

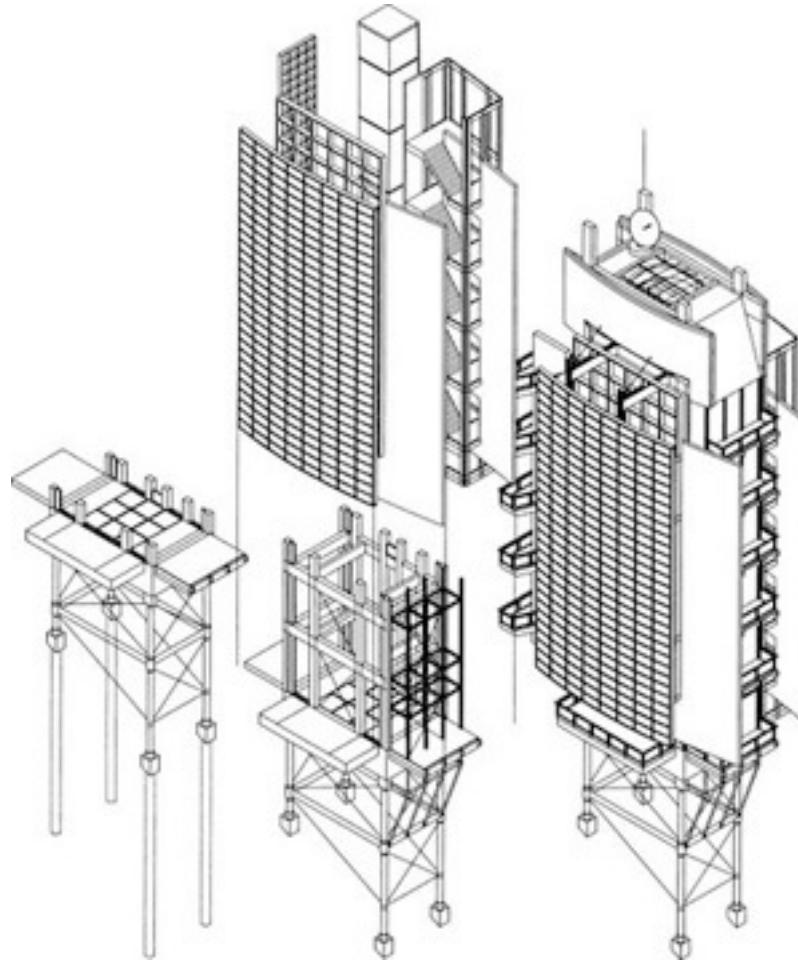
Query database B for activity of user 3...

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.



modwsgi

Python WSGI adapter module for Apache.

[Project Home](#) [Downloads](#) [Wiki](#) [Issues](#) [Source](#)

[Summary](#) [People](#)

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 via the Python [WSGI](#) interface. The module would be suitable for use in hosting high performance production web sites or 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 same process as the normal Apache child processes. WSGI applications when run in this mode will therefore share the same process 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 fashion 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 do not 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 managed hosting arrangements, FASTCGI in particular would still be the preferred choice in most situations.

The (Python) framework world



Web micro-framework battle

<http://www.youtube.com/watch?v=AYjPlMe0BhA>

The (Python) framework world



Flask

web development,
one drop at a time

<http://flask.pocoo.org/>

Hello World

fhello.py

Hello World

fhello.py

```
from flask import Flask
```

Hello World

fhello.py

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

Hello World

fhello.py

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

@app.route("/")
```

Hello World

fhello.py

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

@app.route("/")
def hello():
```

Hello World

fhello.py

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

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

Hello World

fhello.py

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

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

if __name__ == "__main__":
```

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
```

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”

`urls.py`

HTTP Methods

`methods.py`

URL Route Registration

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

```
@app.route('/')
```

urls.py

HTTP Methods

methods.py

URL Route Registration

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

```
@app.route('/')
....
```

urls.py

HTTP Methods

methods.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')
```

urls.py

HTTP Methods

methods.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')
...
```

urls.py

HTTP Methods

methods.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>')
```

urls.py

HTTP Methods

methods.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):
```

urls.py

HTTP Methods

methods.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
```

urls.py

HTTP Methods

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

methods.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
```

urls.py

HTTP Methods

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

methods.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
```

urls.py

HTTP Methods

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

methods.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
```

urls.py

HTTP Methods

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

methods.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
```

urls.py

HTTP Methods

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

methods.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>')
```

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

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):
```

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

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

```
7     """) is allowed to respond to
8     @app.route('/welcome', methods=['GET', 'POST'])
9     def welcome():
10
11         if request.method == 'POST':
12             username = request.form['name']
13             if username not in ("", " ", None):
14                 return "Hey %s, what's up?" % username
15             else:
16                 return """We really want to know your name. Add it
17                         here""" % url_for("welcome")
18         else:
19             ## this is a normal GET request
20             return """
21                 <form action="welcome" method="POST">
22                     What is your name?
23                     <input type="text" name="name" />
24                     <input type="submit" />
25                 </form>"""
26
```

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

Templates

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

```
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/base.html

Templates

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

```
<!DOCTYPE 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()
```

hello.py

templates/base.html

Templates

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

```
<!DOCTYPE html>
<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()
```

hello.py

templates/base.html

Templates

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

```
<!DOCTYPE html>
<html>
  <head>
```

```
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()
```

hello.py

templates/base.html

Templates

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

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

```
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()
```

hello.py

templates/base.html

Templates

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

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

```
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()
```

hello.py

templates/base.html

Templates

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

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

```
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()
```

hello.py

templates/base.html

Templates

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

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

  <body>
```

```
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()
```

hello.py

templates/base.html

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>
```

```
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()
```

hello.py

templates/base.html

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>
```

```
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()
```

hello.py

templates/base.html

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 %}
    
```

```
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()
```

theHello.py

templates/base.html

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>
```

```
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()
```

theHello.py

templates/base.html

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>
```

```
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/base.html

Templates

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

```
10
11 @app.route('/welcome', methods=['GET', 'POST'])
12 def welcome():
13
14     if request.method == 'POST':
15         username = request.form['name']
16         if username not in ("", " ", None):
17             return "Hey %s, what's up?" % username
18         else:
19             return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21     else:
22         ## this is a normal GET request
23         return render_template("form.html")
24
25 if __name__ == "__main__":
26     app.run()
27
```

thello.py

templates/form.html

Templates

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

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

```
10
11 @app.route('/welcome', methods=['GET', 'POST'])
12 def welcome():
13
14     if request.method == 'POST':
15         username = request.form['name']
16         if username not in ("", " ", None):
17             return "Hey %s, what's up?" % username
18         else:
19             return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21     else:
22         ## this is a normal GET request
23         return render_template("form.html")
24
25 if __name__ == "__main__":
26     app.run()
27
```

theHello.py

templates/form.html

Templates

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

```
{% extends "base.html" %}  
{% set page_title = 'My Form' %}
```

```
10  
11 @app.route('/welcome', methods=['GET', 'POST'])  
12 def welcome():  
13  
14     if request.method == 'POST':  
15         username = request.form['name']  
16         if username not in ("", " ", None):  
17             return "Hey %s, what's up?" % username  
18         else:  
19             return """We really want to know your name!  
20             <a href='%s'>here</a>"""  
21     else:  
22         ## this is a normal GET request  
23         return render_template("form.html")  
24  
25 if __name__ == "__main__":  
26     app.run()  
27
```

thello.py

templates/form.html

Templates

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

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

{% set page_title = 'My Form' %}

{% block content %}
```

```
10
11 @app.route('/welcome', methods=['GET', 'POST'])
12 def welcome():
13
14     if request.method == 'POST':
15         username = request.form['name']
16         if username not in ("", " ", None):
17             return "Hey %s, what's up?" % username
18         else:
19             return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21     else:
22         ## this is a normal GET request
23         return render_template("form.html")
24
25 if __name__ == "__main__":
26     app.run()
27
```

theHello.py

templates/form.html

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">
```

```
10
11     @app.route('/welcome', methods=['GET', 'POST'])
12     def welcome():
13
14         if request.method == 'POST':
15             username = request.form['name']
16             if username not in ("", " ", None):
17                 return "Hey %s, what's up?" % username
18             else:
19                 return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21         else:
22             ## this is a normal GET request
23             return render_template("form.html")
24
25     if __name__ == "__main__":
26         app.run()
27
```

theHello.py

templates/form.html

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?

```

```
10
11     @app.route('/welcome', methods=['GET', 'POST'])
12     def welcome():
13
14         if request.method == 'POST':
15             username = request.form['name']
16             if username not in ("", " ", None):
17                 return "Hey %s, what's up?" % username
18             else:
19                 return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21         else:
22             ## this is a normal GET request
23             return render_template("form.html")
24
25     if __name__ == "__main__":
26         app.run()
27
```

theHello.py

templates/form.html

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" />
```

```
10
11     @app.route('/welcome', methods=['GET', 'POST'])
12     def welcome():
13
14         if request.method == 'POST':
15             username = request.form['name']
16             if username not in ("", " ", None):
17                 return "Hey %s, what's up?" % username
18             else:
19                 return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21         else:
22             ## this is a normal GET request
23             return render_template("form.html")
24
25     if __name__ == "__main__":
26         app.run()
27
```

theHello.py

templates/form.html

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" />
```

```
10
11     @app.route('/welcome', methods=['GET', 'POST'])
12     def welcome():
13
14         if request.method == 'POST':
15             username = request.form['name']
16             if username not in ("", " ", None):
17                 return "Hey %s, what's up?" % username
18             else:
19                 return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21         else:
22             ## this is a normal GET request
23             return render_template("form.html")
24
25     if __name__ == "__main__":
26         app.run()
27
```

theHello.py

templates/form.html

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>
```

```
10
11 @app.route('/welcome', methods=['GET', 'POST'])
12 def welcome():
13
14     if request.method == 'POST':
15         username = request.form['name']
16         if username not in ("", " ", None):
17             return "Hey %s, what's up?" % username
18         else:
19             return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21     else:
22         ## this is a normal GET request
23         return render_template("form.html")
24
25     if __name__ == "__main__":
26         app.run()
27
```

theHello.py

templates/form.html

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 %}
```

```
10
11 @app.route('/welcome', methods=['GET', 'POST'])
12 def welcome():
13
14     if request.method == 'POST':
15         username = request.form['name']
16         if username not in ("", " ", None):
17             return "Hey %s, what's up?" % username
18         else:
19             return """We really want to know your name!
20                         <a href='%s'>here</a>"""
21     else:
22         ## this is a normal GET request
23         return render_template("form.html")
24
25     if __name__ == "__main__":
26         app.run()
27
```

theHello.py

templates/form.html

flask uses Jinja2 templating



A screenshot of a web browser window. The address bar shows the URL <https://jinja.pocoo.org/docs/templates/>. The page content is visible below the header.



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.

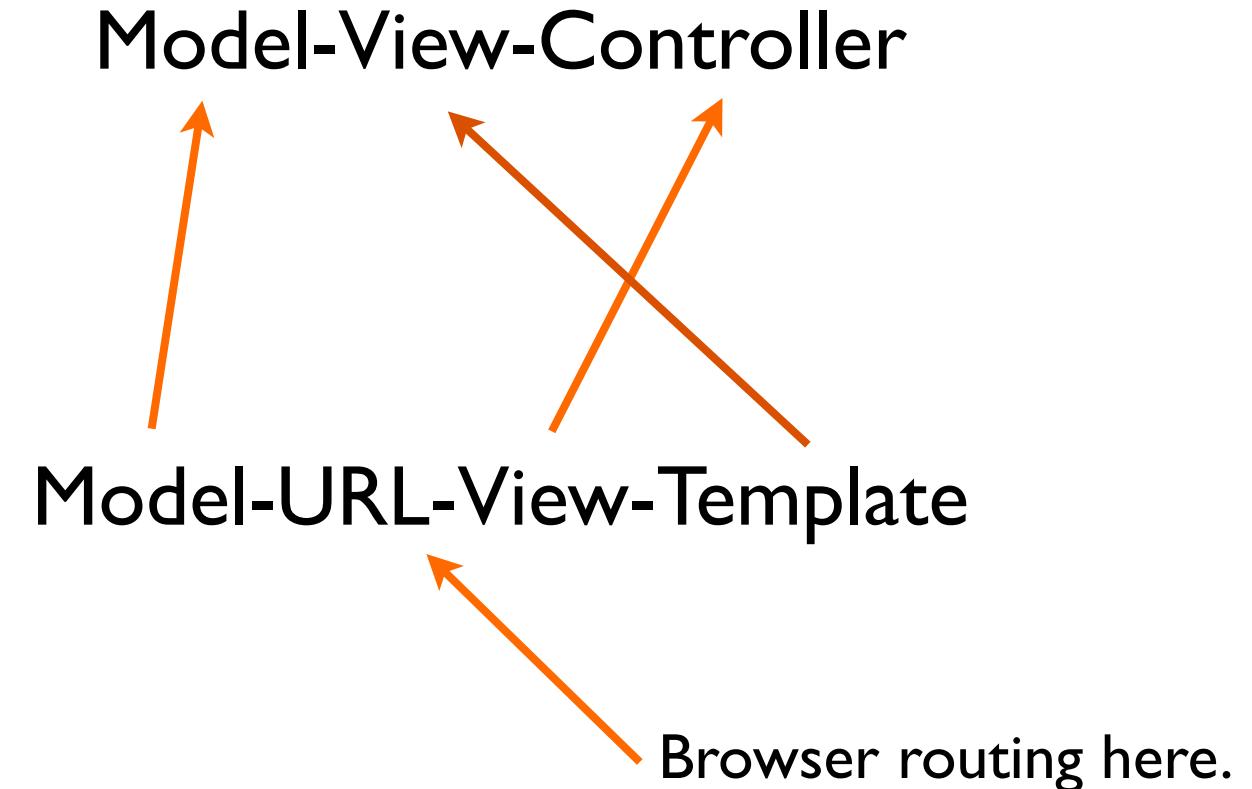
A template contains **variables** or **expressions**, which get replaced with values when the template is evaluated, and tags, which control the logic of the template. The template syntax is heavily inspired by Django and Python.

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...

Remember?



Flask can 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.

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
```

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
```

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__)
```

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'
```

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)
```

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):
```

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)
```

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)
```

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)
```

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):
```

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
```

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.

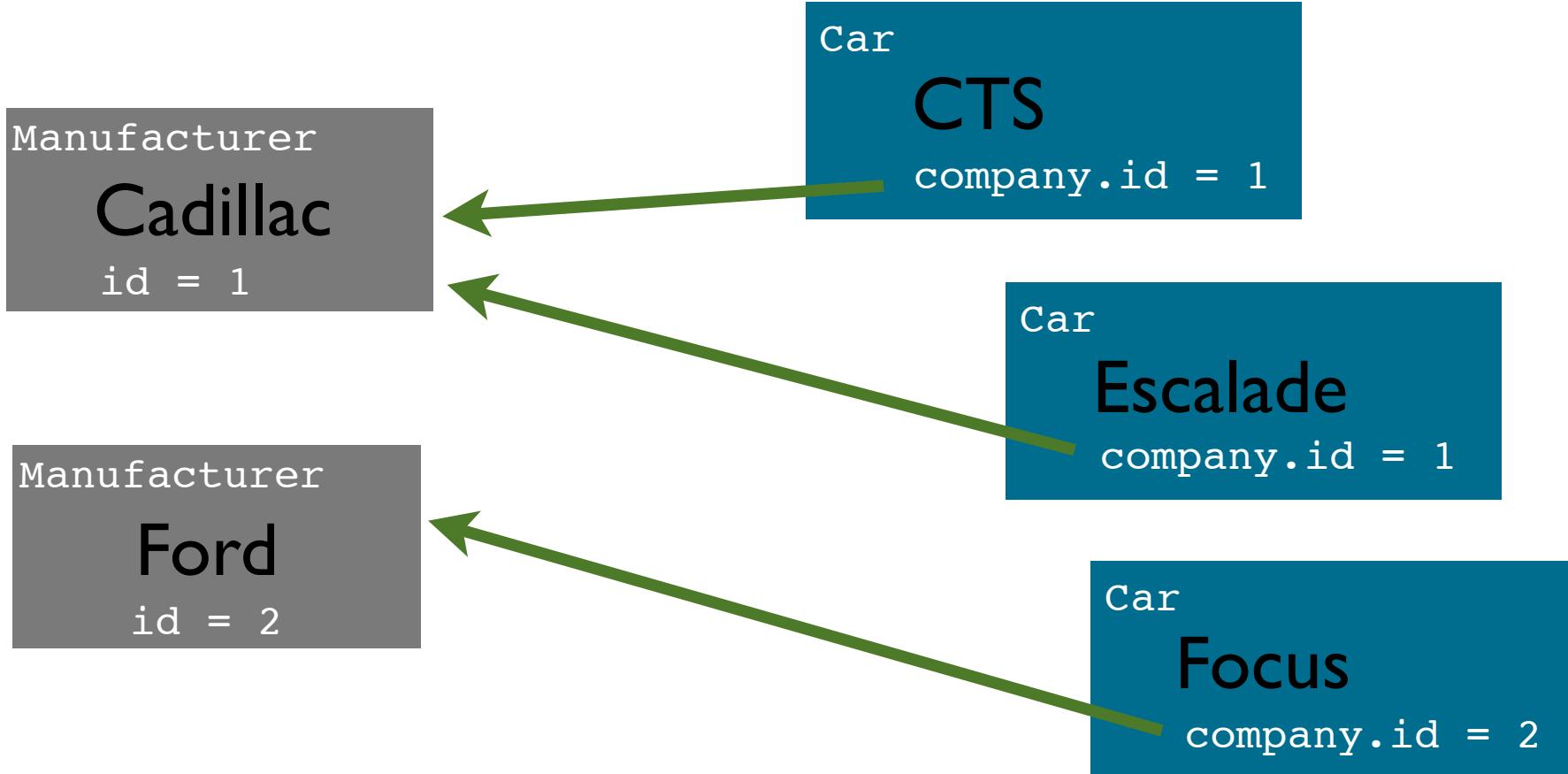
```
from flask import Flask
from flask.ext.sqlalchemy import SQLAlchemy

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



Model Relationships

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

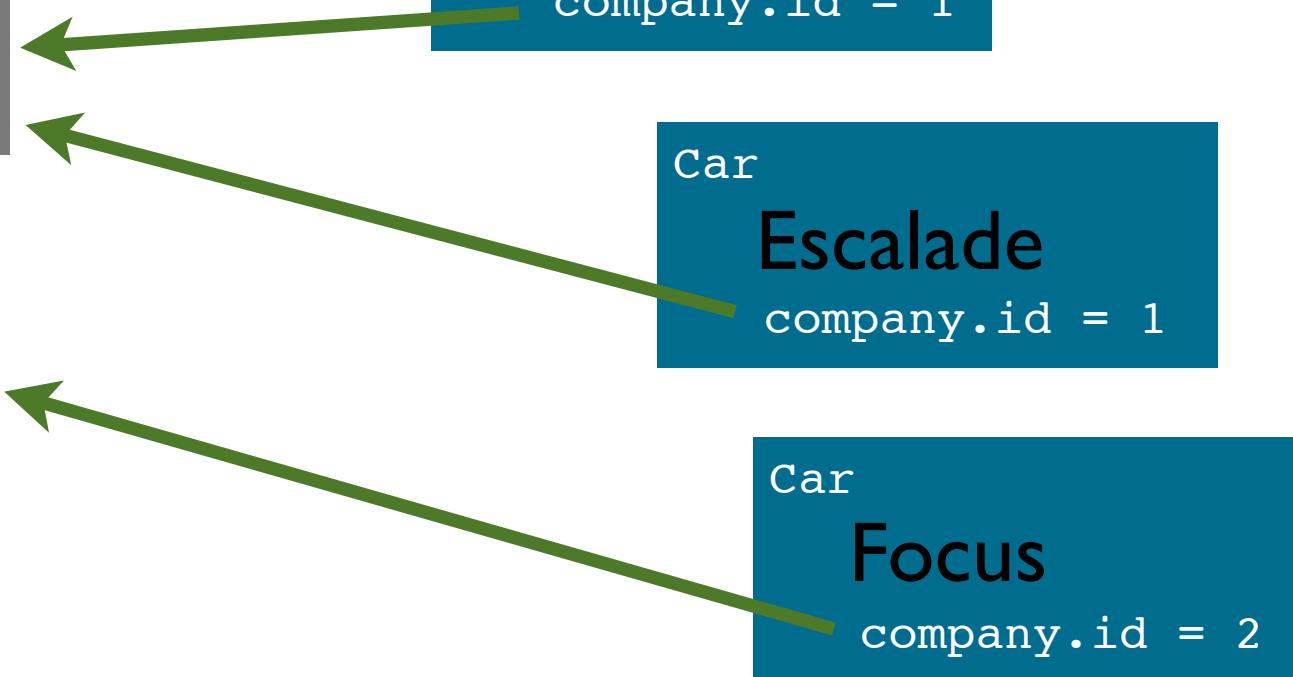
Manufacturer
Cadillac
id = 1

Manufacturer
Ford
id = 2

Car
CTS
company.id = 1

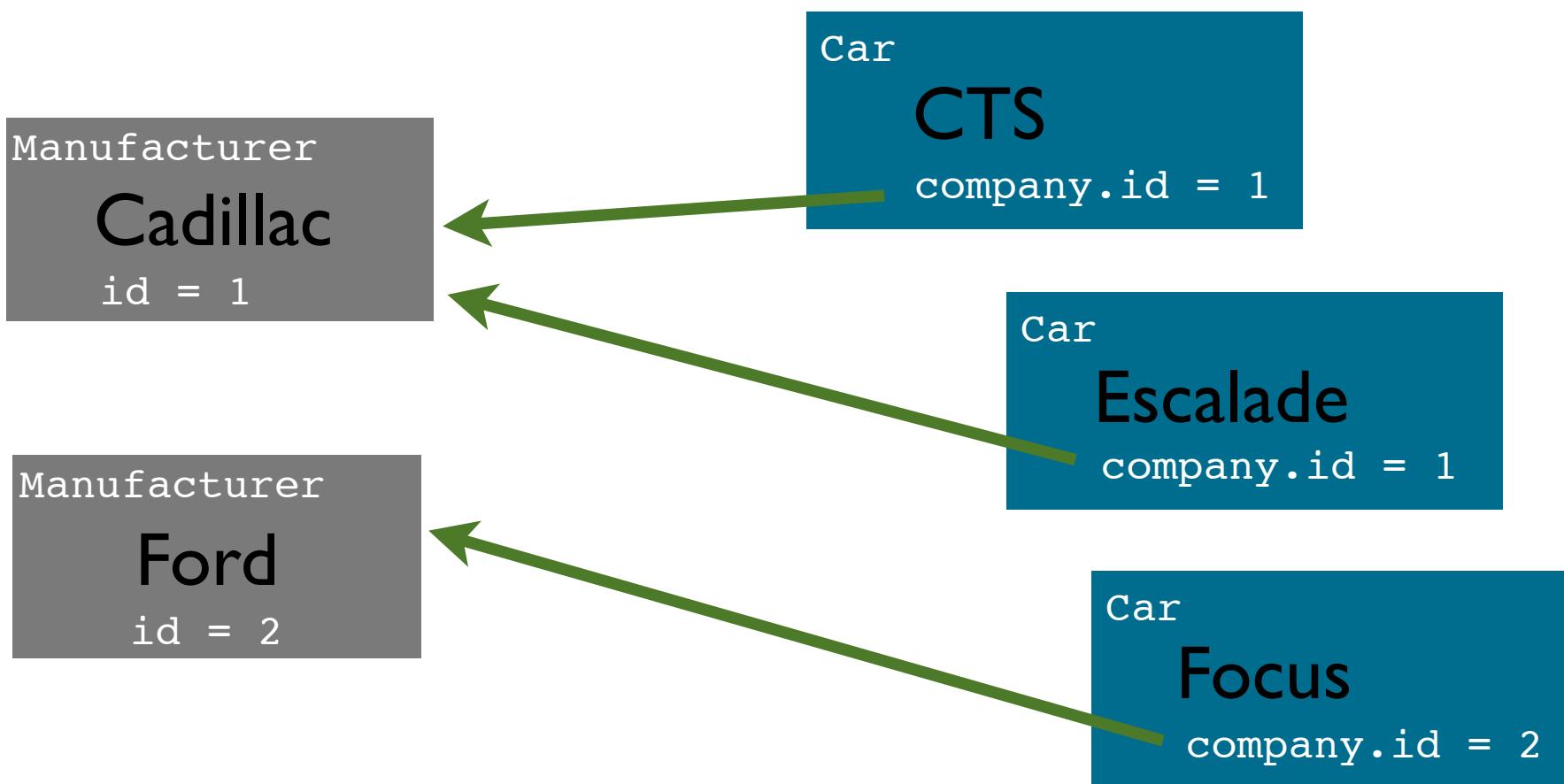
Car
Escalade
company.id = 1

Car
Focus
company.id = 2



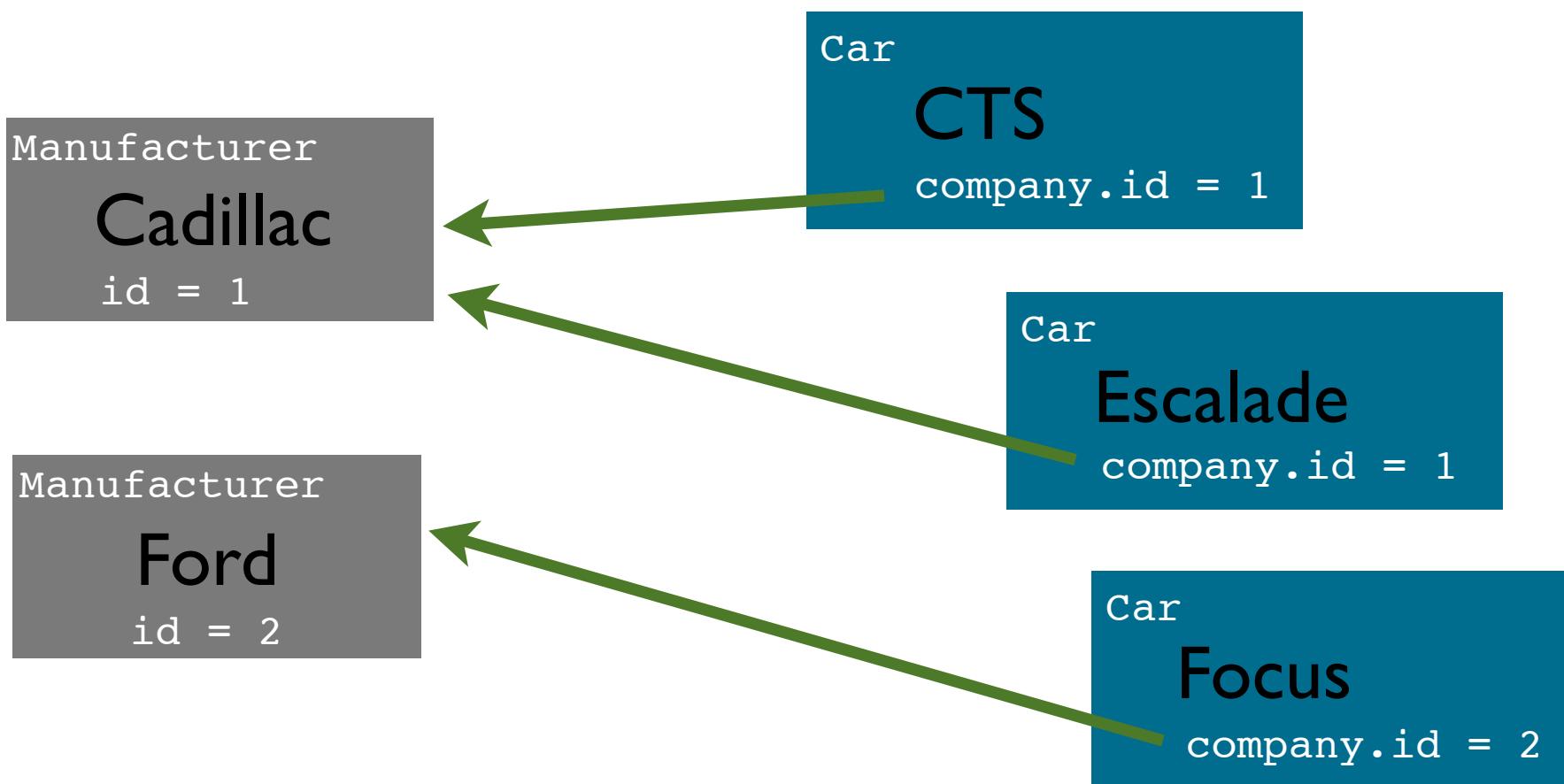
Model Relationships

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



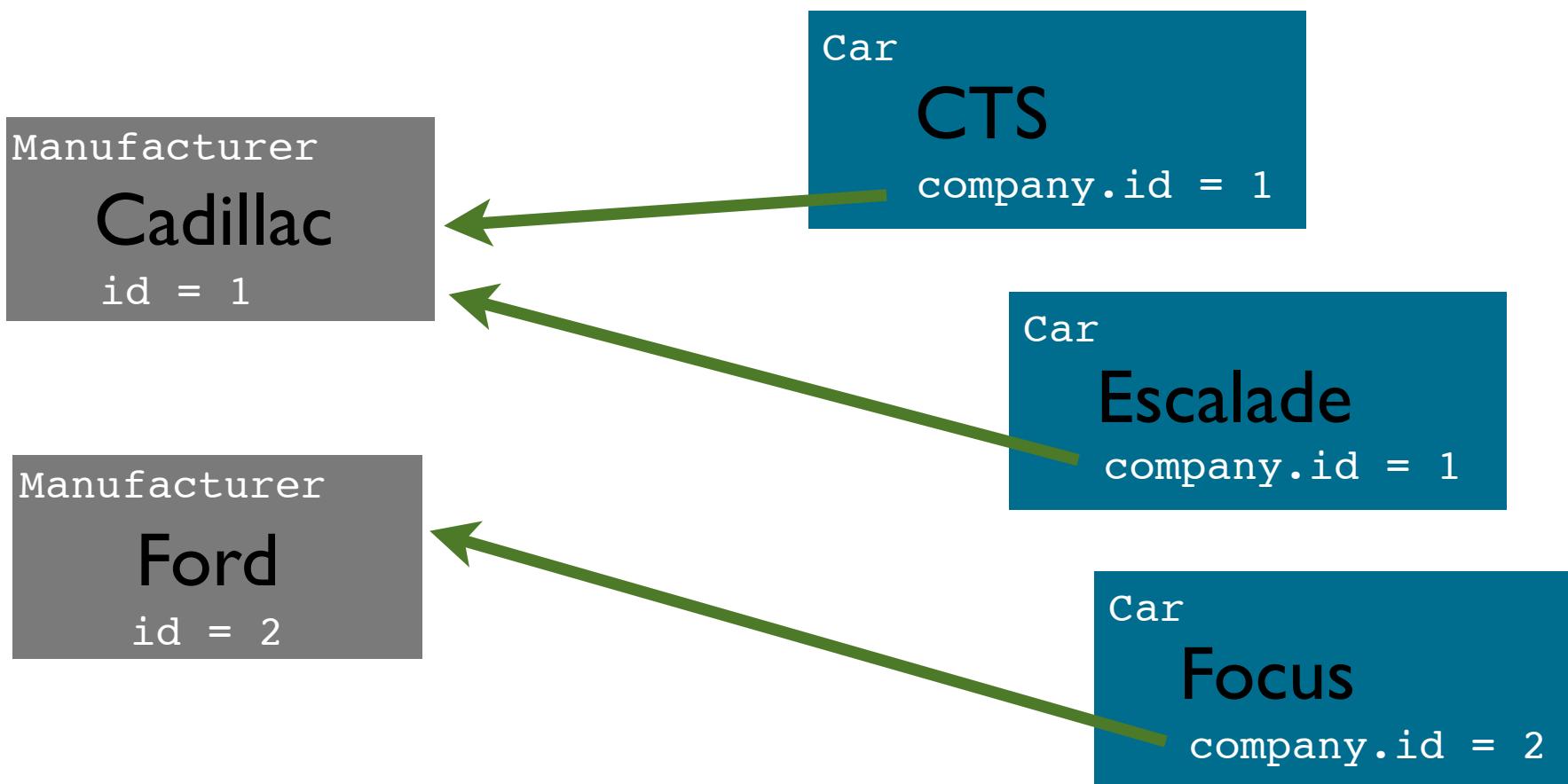
Model Relationships

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

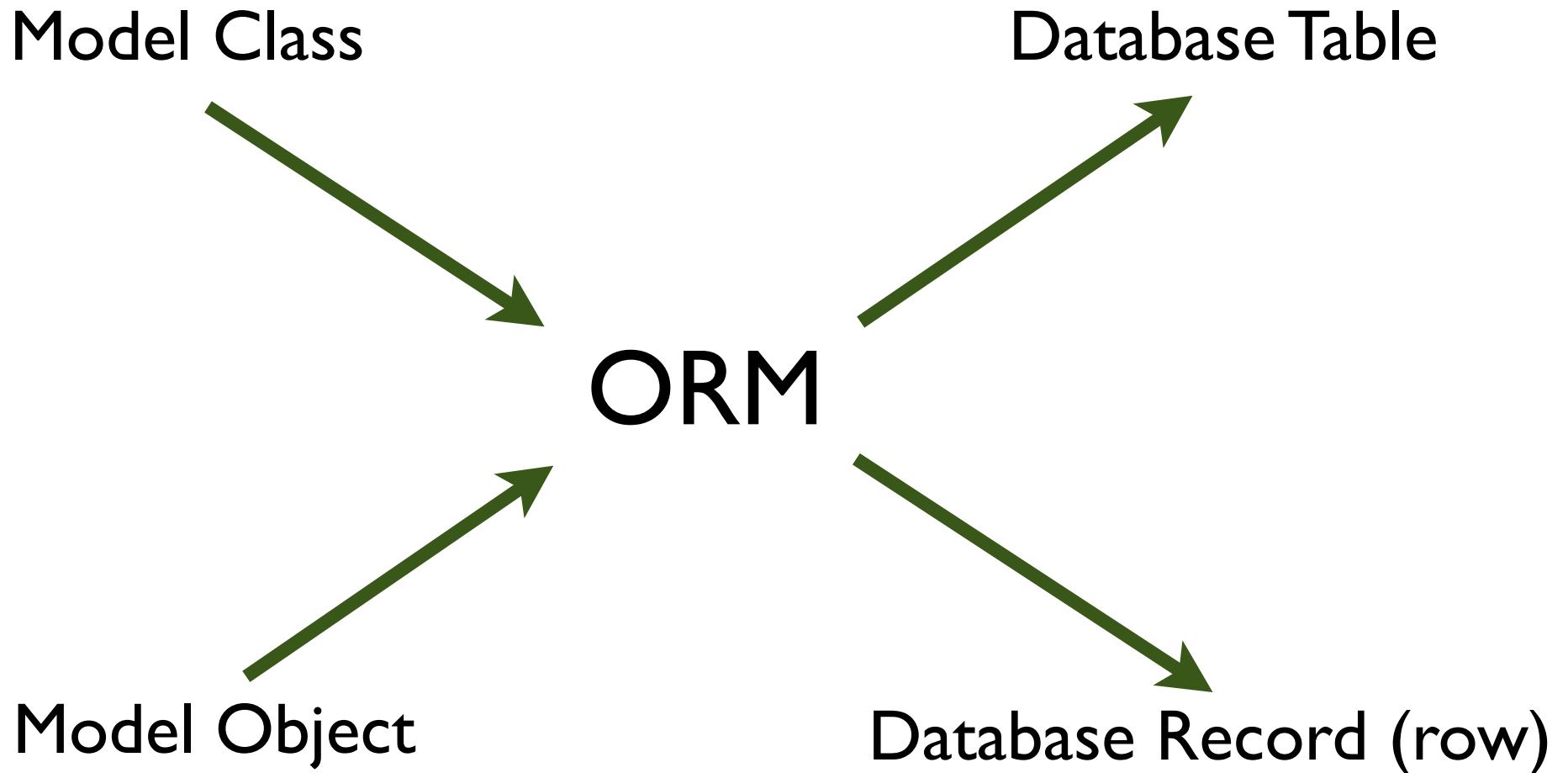


Model Relationships

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



Models



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)*

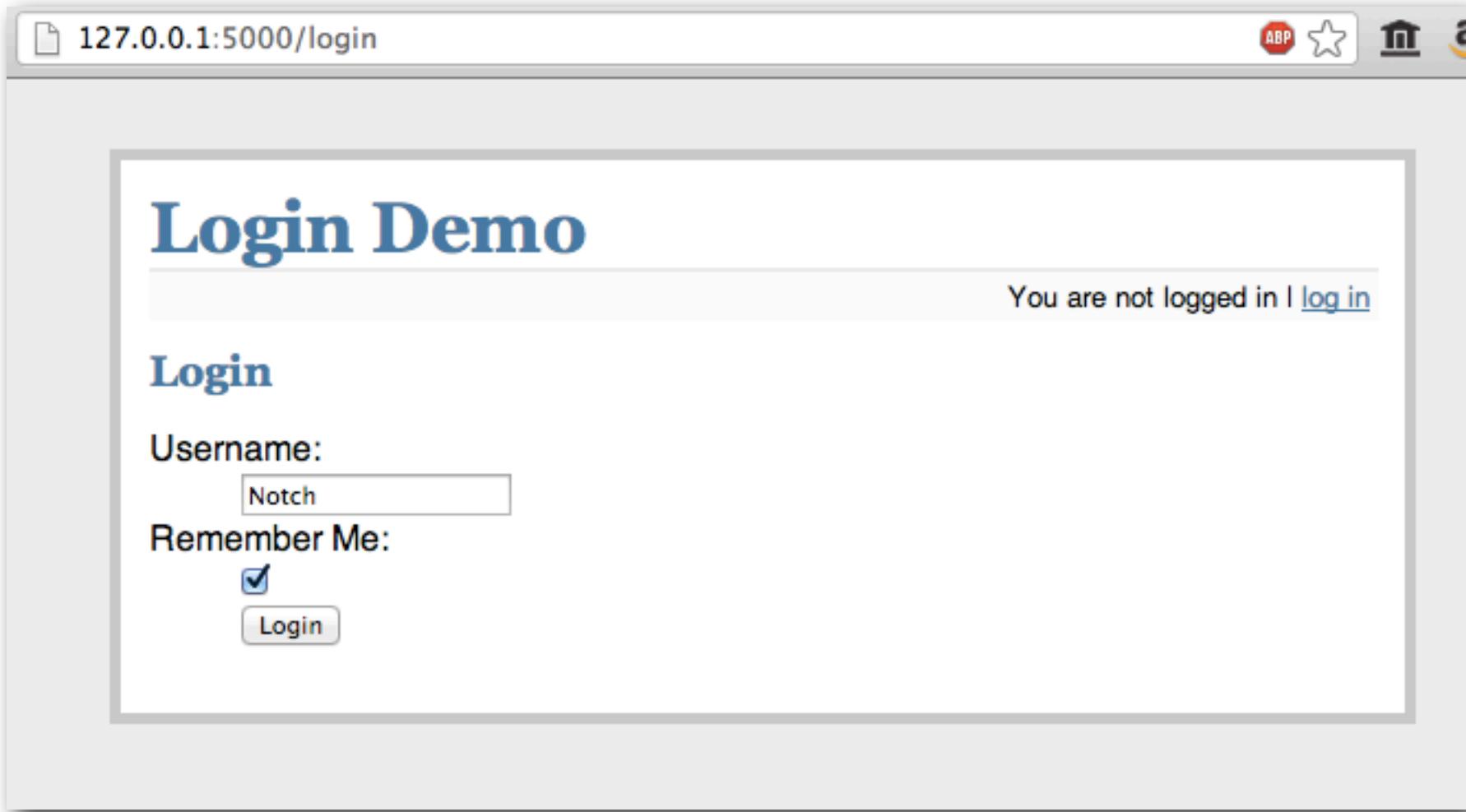
OpenID

*Sending Mail
(Flask-Mail)*

Themes

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

```
pip install flask-login
```



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

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?



Google code

e.g. "templates" or "datastore"

 Search

★ 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



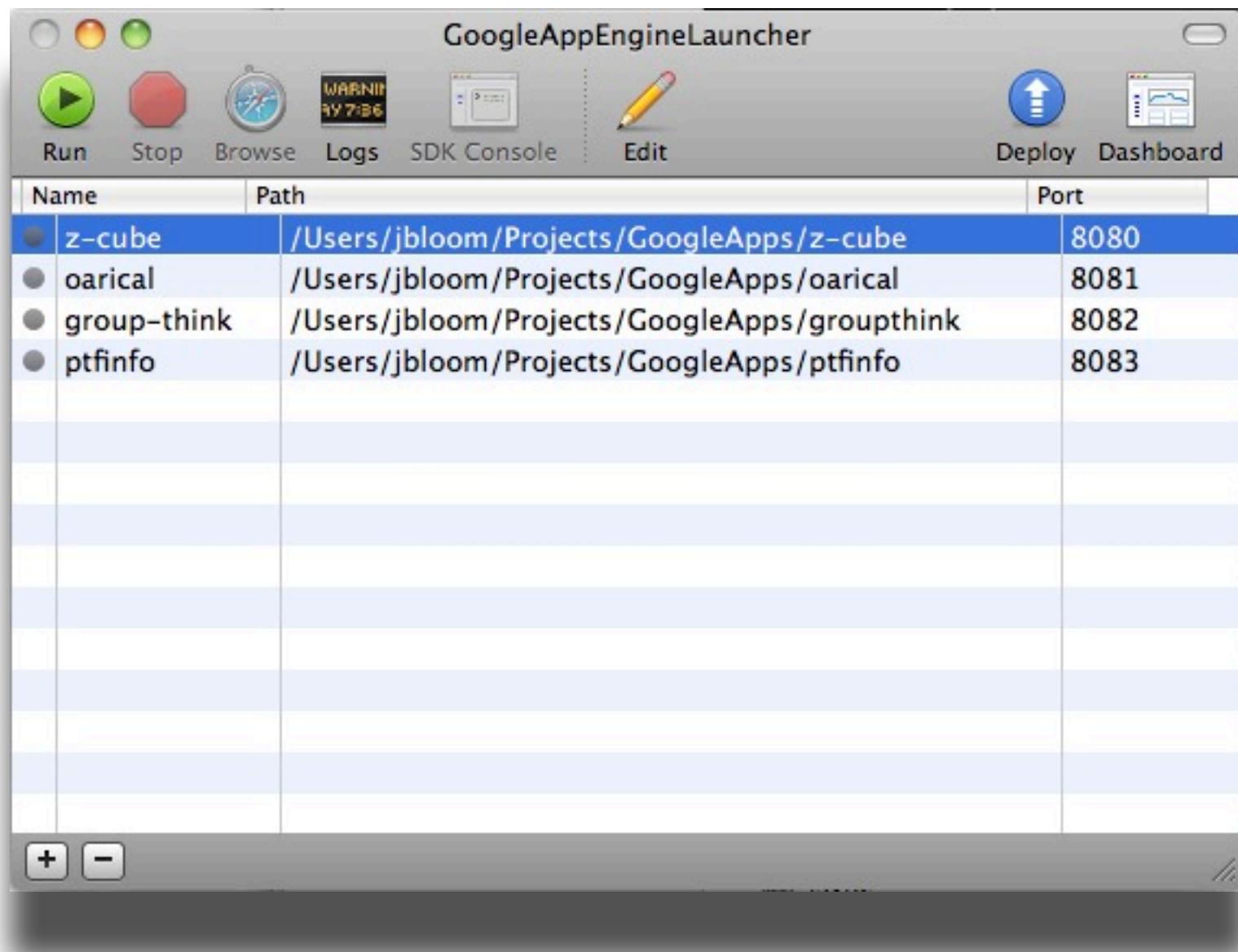
Developing and deploying
on Google App Engine.
[Watch Now](#)



[links from AppEngine.reddit.com](#)

▲ New App Engine SDK 1.3.8 includes

SDK allows you to run a local version of your projects



* app.yaml * index.yaml * main.py

```
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 = "<a href='%s'>Click here to get started as a site user</a>" % "/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))~
```

z-cube
app.yaml
app.yaml
index.yaml
main.py
static
templates
ttt.html
ttt.html~

online management...

The screenshot shows a web browser window for the Google App Engine Data Viewer. The URL in the address bar is `appengine.google.com/datastore/explorer?submitted=1&app_id=z-cube&kind=Device...`. The browser's toolbar includes icons for Vox Charta, iWork.com - web_dev-2, and Data Viewer - z-cube. Below the toolbar, the menu bar shows links for Apple, Yahoo!, Google Maps, YouTube, Wikipedia, Popular, News, and bit.ly sidebar. On the right side of the menu bar, there are icons for Other Books and Help.

The main content area displays the Data Viewer interface for the 'z-cube' application. The title bar says 'Google app engine'. The top navigation bar includes a dropdown for 'z-cube' and 'Version: 1', along with links for 'My Application' and 'Sign Out'.

The left sidebar has sections for Main (Dashboard, Quota Details, Instances, Logs, Cron Jobs, Task Queues, Blacklist), Data (Datastore Indexes, Datastore Viewer, Datastore Statistics, Blob Viewer, Datastore Admin), and Administration (Application Settings). The 'Datastore Viewer' link under 'Data' is currently selected.

The central part of the page shows a query interface with tabs for 'Query' (selected) and 'Create'. It specifies 'By kind: DeviceUsers' and shows 'kinds as of 0:00:05 ago'. There is also a '+ Options' link.

The main content area is titled 'DeviceUsers Entities in Empty Namespace' and lists two entities:

ID/Name	device	user
id=1002	agZ6LWN1YmVyDQsSBkRldmljZRjpBww Device: id=1001	agZ6LWN1YmVyDgsSCFnpdGVVc2VyGAEM SiteUser: id=1

Below the table is a 'Delete' button. Navigation links at the bottom include '< Prev 20 1-1 Next 20 >' and '< Prev 20 1-1 Next 20 >' on the right.