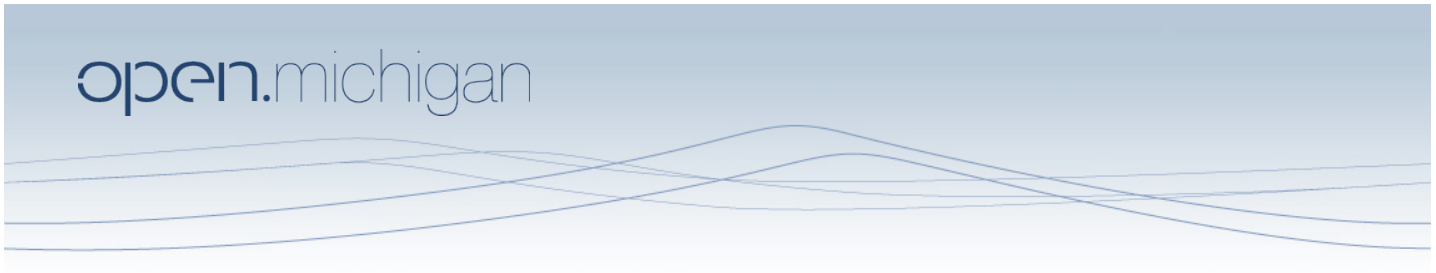
App Engine
Web App Framework

Jim Eng / Charles Severance
jimeng@umich.edu / csev@umich.edu

www.appenginelearn.com

Textbook: Using Google App Engine, Charles Severance (Chapter 5)



Internet

HTML JavaScript

AJAX CSS

HTTP Request

Response GET

POST

Python Data Store

Templates memcache

WebApp

The webapp Framework

- While we could write our application using the low-level data provided to our Python code, this would become very tedious
- We would constantly be reading a lot of Internet Standards documents

```
Environment keys:

HTTP_COOKIE : camtoolspref=
SERVER_SOFTWARE : Development/1.0
SCRIPT_NAME :
REQUEST_METHOD : GET
PATH_INFO : /
SERVER_PROTOCOL : HTTP/1.0
QUERY_STRING :
CONTENT_LENGTH :
HTTP_USER_AGENT : Mozilla/5.0 (Macintosh; U; Inte
HTTP_CONNECTION : keep-alive
SERVER_NAME : localhost
REMOTE_ADDR : 127.0.0.1
PATH_TRANSLATED : /Users/csev/Desktop/teach/a539-
SERVER_PORT : 8081
AUTH_DOMAIN : gmail.com
CURRENT_VERSION_ID : 1.1
HTTP_HOST : localhost:8081
TZ : UTC
HTTP_CACHE_CONTROL : max-age=0
USER_EMAIL :
HTTP_ACCEPT : text/xml,application/xml,applicatio
APPLICATION_ID : ae-02-dumper
GATEWAY_INTERFACE : CGI/1.1
HTTP_ACCEPT_LANGUAGE : en-us
CONTENT_TYPE : application/x-www-form-urlencoded
HTTP_ACCEPT_ENCODING : gzip, deflate
```

The webapp Framework

- Someone has already written the common code that knows all the details of HTTP (HyperText Transport Protocol)
- We just import it and then use it.

```
import wsgiref.handlers
from google.appengine.ext import webapp
```

import wsgiref.handlers

- http://docs.python.org/library/wsgiref.html

`wsgiref` — WSGI Utilities and Reference Implementation

New in version 2.5.

The Web Server Gateway Interface (WSGI) is a standard interface between web server software and web applications written in Python. Having a standard interface makes it easy to use an application that supports WSGI with a number of different web servers.

Only authors of web servers and programming frameworks need to know every detail and corner case of the WSGI design. You don't need to understand every detail of WSGI just to install a WSGI application or to write a web application using an existing framework.

`wsgiref` is a reference implementation of the WSGI specification that can be used to add WSGI support to a web server or framework. It provides utilities for manipulating WSGI environment variables and response headers, base classes for implementing WSGI servers, a demo HTTP server that serves WSGI applications, and a validation tool that checks WSGI servers and applications for conformance to the WSGI specification (**PEP 333**).

See http://www.wsgi.org for more information about WSGI, and links to tutorials and other resources.

import wsgiref.handlers

- http://docs.python.org/library/wsgiref.html

`wsgiref.handlers` – server/gateway base classes

This module provides base handler classes for implementing WSGI servers and gateways. These base classes handle most of the work of communicating with a WSGI application, as long as they are given a CGI-like environment, along with input, output, and error streams.

class `wsgiref.handlers.`**`CGIHandler`**

CGI-based invocation via `sys.stdin`, `sys.stdout`, `sys.stderr` and `os.environ`. This is useful when you have a WSGI application and want to run it as a CGI script. Simply invoke `CGIHandler().run(app)`, where `app` is the WSGI application object you wish to invoke.

from google.appengine.ext import webapp

- http://code.google.com/appengine/docs/python/gettingstarted/usingwebapp.html

Using the webapp Framework

The CGI standard is simple, but it would be cumbersome to write all of the code that uses it by hand. Web application frameworks handle these details for you, so you can focus your development efforts on your application's features. Google App Engine supports any framework written in pure Python that speaks CGI (and any WSGI-compliant framework using a CGI adaptor), including Django, CherryPy, Pylons, and web.py. You can bundle a framework of your choosing with your application code by copying its code into your application directory.

App Engine includes a simple web application framework of its own, called `webapp`. The `webapp` framework is already installed in the App Engine environment and in the SDK, so you do not need to bundle it with your application code to use it. We will use `webapp` for the rest of this tutorial.

Hello, webapp!

A `webapp` application has three parts:

- one or more `RequestHandler` classes that process requests and build responses
- a `WSGIApplication` instance that routes incoming requests to handlers based on the URL
- a main routine that runs the `WSGIApplication` using a CGI adaptor

Let's rewrite our friendly greeting as a `webapp` application. Edit `helloworld/helloworld.py` and replace its contents with the following:

```
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app

class MainPage(webapp.RequestHandler):
  def get(self):
    self.response.headers['Content-Type'] = 'text/plain'
```

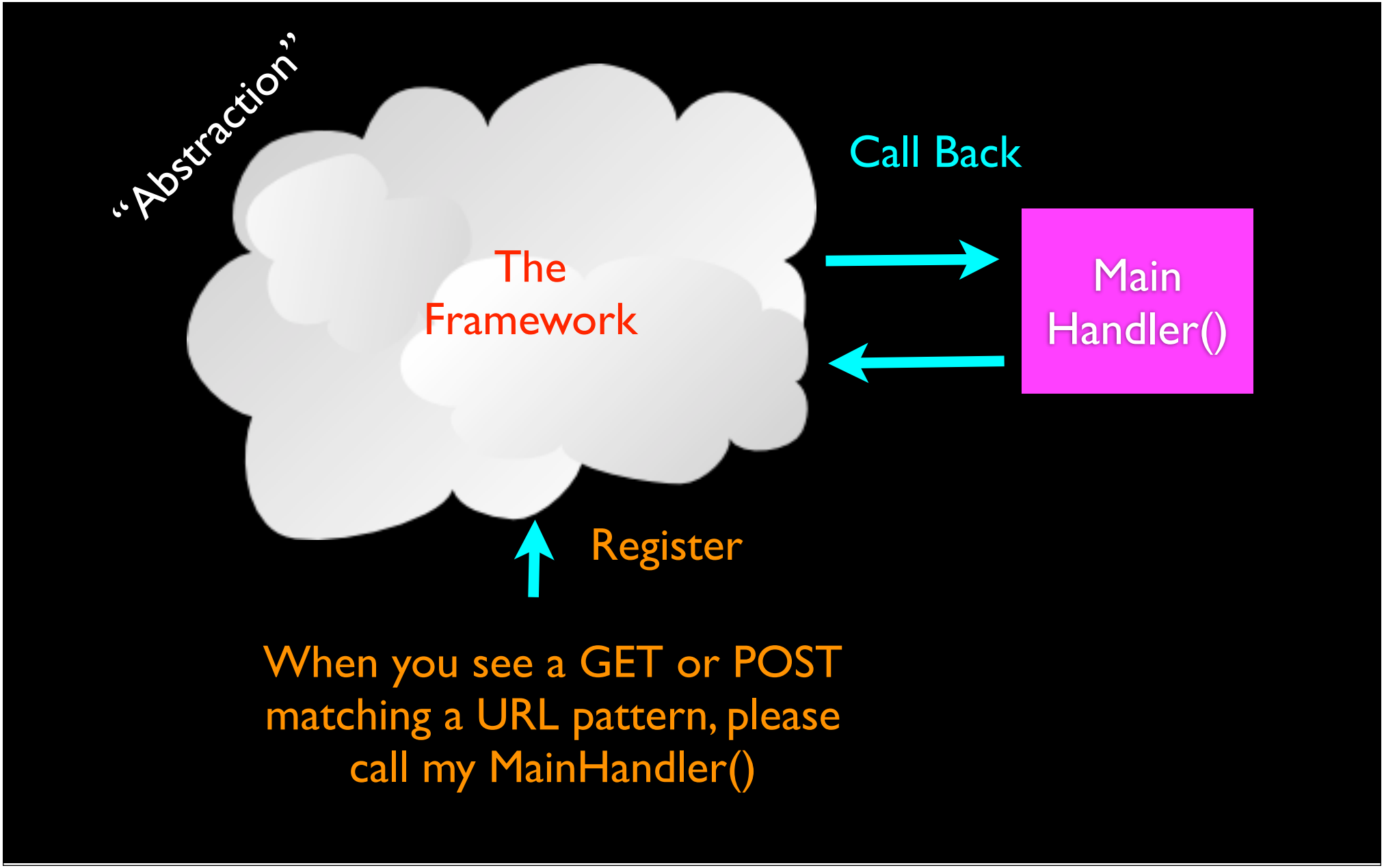
Starting the Framework

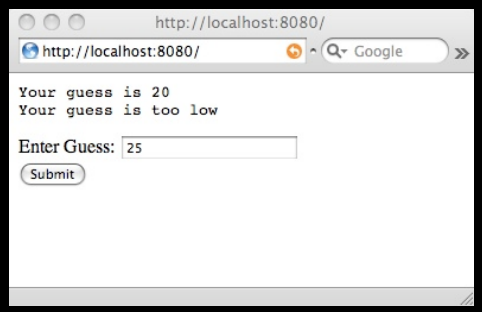
- Define our application and the routing of input URLs to "Handlers"
- Starting the framework to process the current request

```
def main():
    application = webapp.WSGIApplication(
        [('/.*', MainHandler)],
        debug=True)
    wsgiref.handlers.CGIHandler().run(application)
```

What is a Handler?

- When we are dealing with a framework - at times the framework needs to ask us a question or involve us with some bit of processing.
- Often this is called "event processing" or "event handling"
- Another word for this is "callbacks"
- We register interest in certain actions and then when those actions happen - we get called.


"Abstraction"
The Framework
Call Back
Main Handler()
Register
When you see a GET or POST matching a URL pattern, please call my MainHandler()

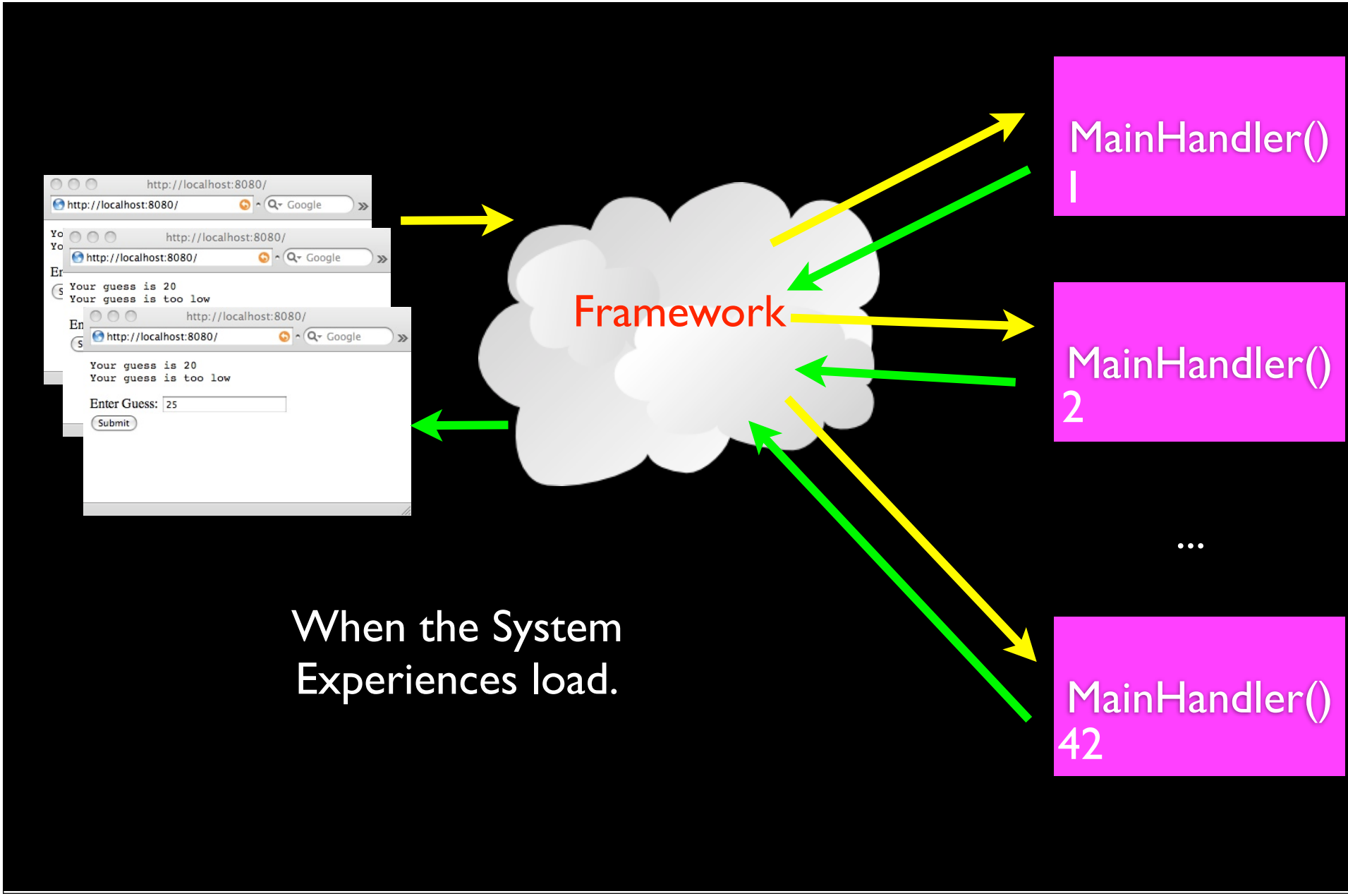


Framework

MainHandler()

Register

When you see a GET or POST matching a URL pattern, please call my MainHandler()

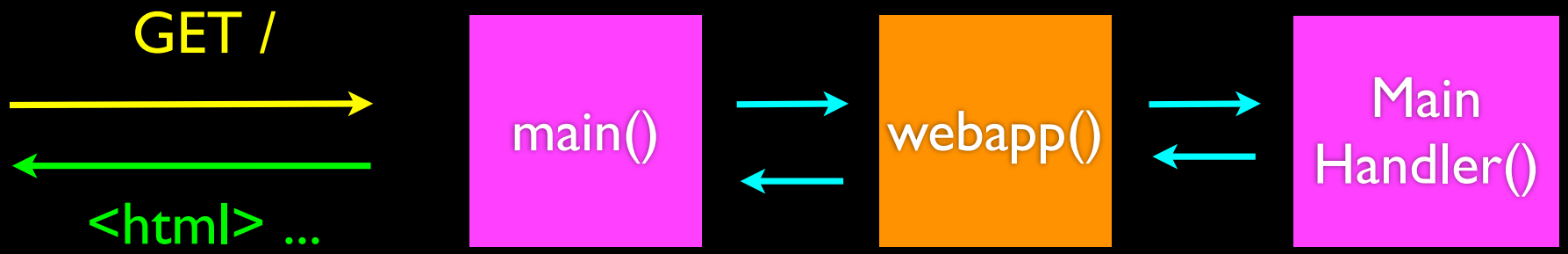


When the System Experiences load.

Starting the Framework

- Sometimes we start the framework - and sometimes it starts us
- In this example - we are starting the framework and giving it an initial configuration

```
def main():
    application = webapp.WSGIApplication(
        [('/.*', MainHandler)],
        debug=True)
    wsgiref.handlers.CGIHandler().run(application)
```



Our main program starts the framework and passes it an initial list of URL routes and the name of the handler code for each route.

```
def main():
    application = webapp.WSGIApplication([
        ('/.*', MainHandler)],debug=True)
    wsgiref.handlers.CGIHandler().run(application)
```

our code

framework

Review: app.yaml

The app.yaml file routes requests amongst different Python scripts. With a particular script, the URL list routes requests amongst handlers.

```
application: ae-03-webapp
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
  script: index.py
```

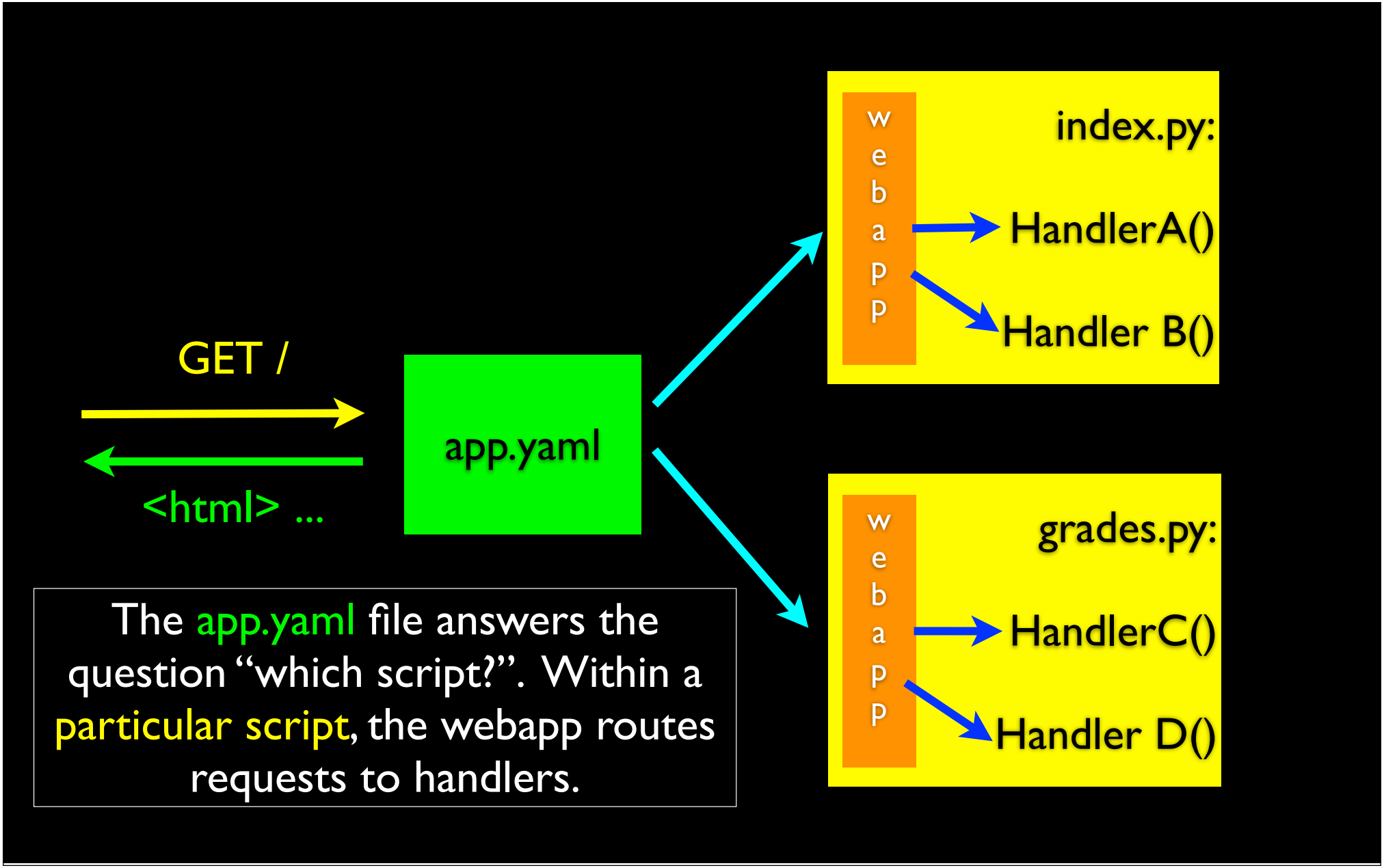
```
def main():
    application = webapp.WSGIApplication(
        [('/.*', MainHandler)],
        debug=True)
    wsgiref.handlers.CGIHandler().run(application)
```

Review: app.yaml

You route URLs in the app.yaml file *and* in the web application framework. For our simple application we simply route all URLs (/.*) to the same place both in app.yaml and in index.py.

```
application: ae-03-webapp
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
  script: index.py
```

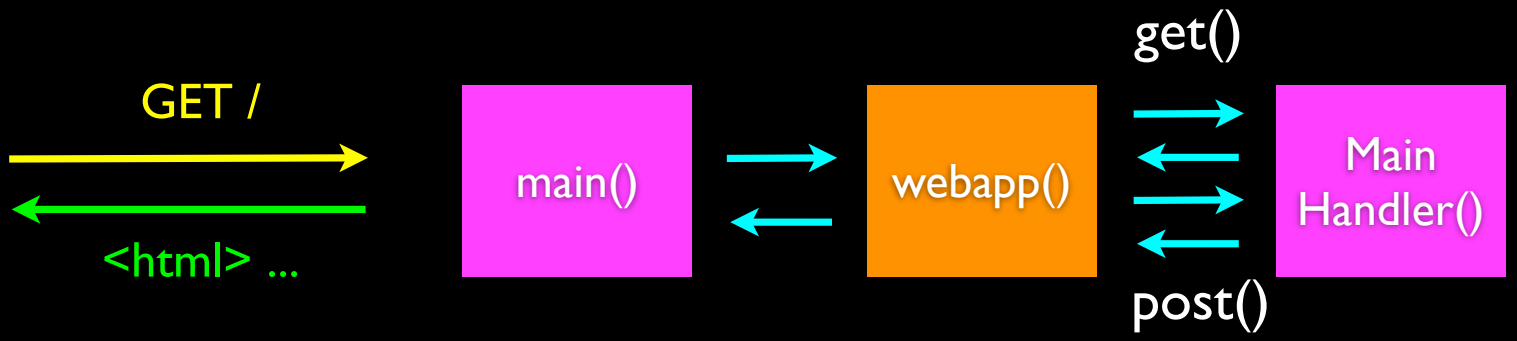
```
def main():
    application = webapp.WSGIApplication([
        ('/.*', MainHandler)],debug=True)
    wsgiref.handlers.CGIHandler().run(application)
```


GET /
<html> ...
app.yaml
webapp
index.py:
HandlerA()
Handler B()
webapp
grades.py:
HandlerC()
Handler D()
The app.yaml file answers the question "which script?". Within a particular script, the webapp routes requests to handlers.

Looking at a Handler

Inside a Handler

- The purpose of a handler is to respond when the framework "needs some help"
- We put methods in the handler for get() and post()



A Pointless Handler

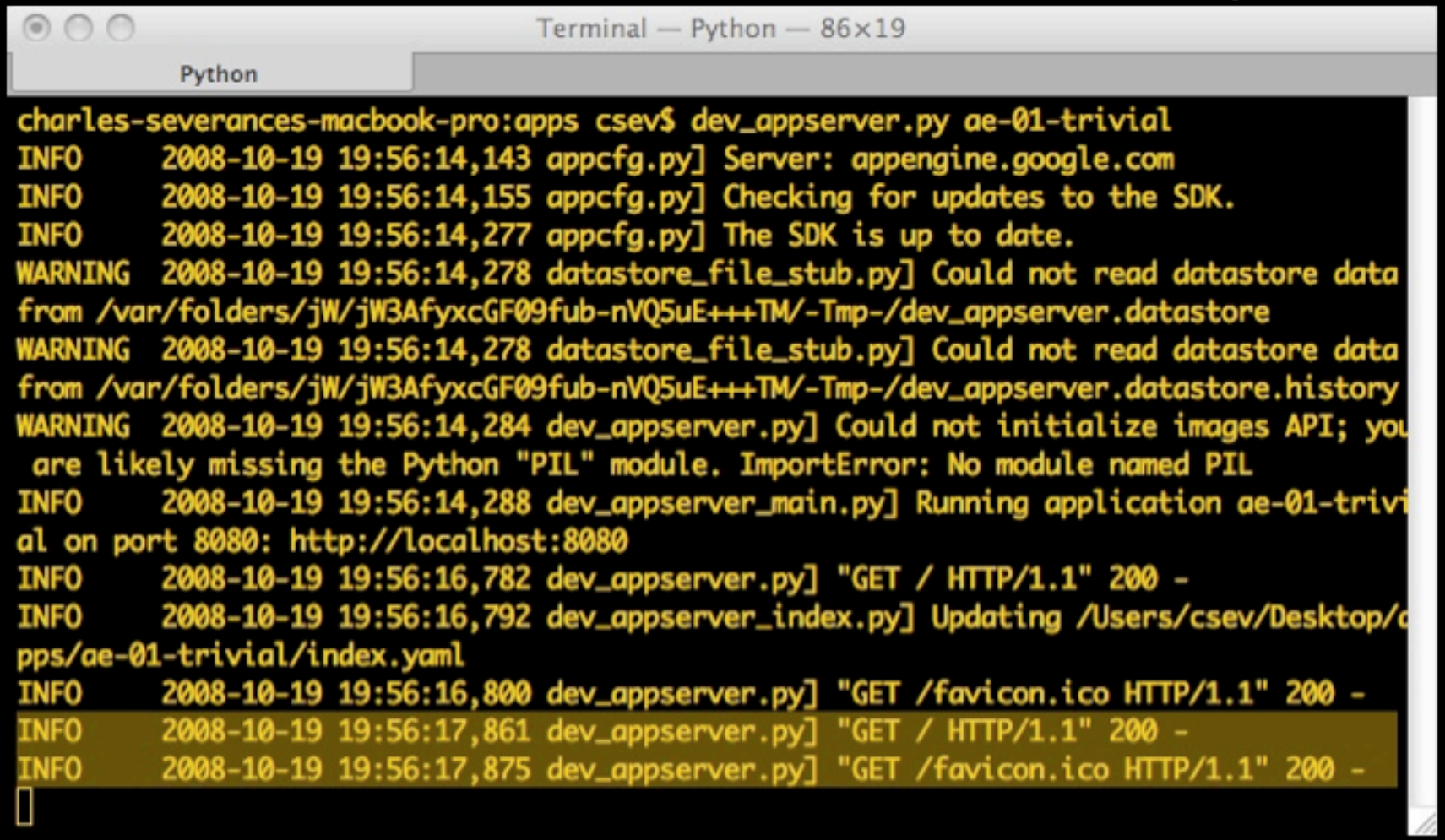
```
class PointlessHandler(webapp.RequestHandler):

  def get(self):
    logging.info("Hello GET")

  def post(self):
    logging.info("Hello POST")
```

This handler, handles a GET and POST request and then does not do anything particularly useful. The post() and get() methods are the contact points between the webapp framework and our code.

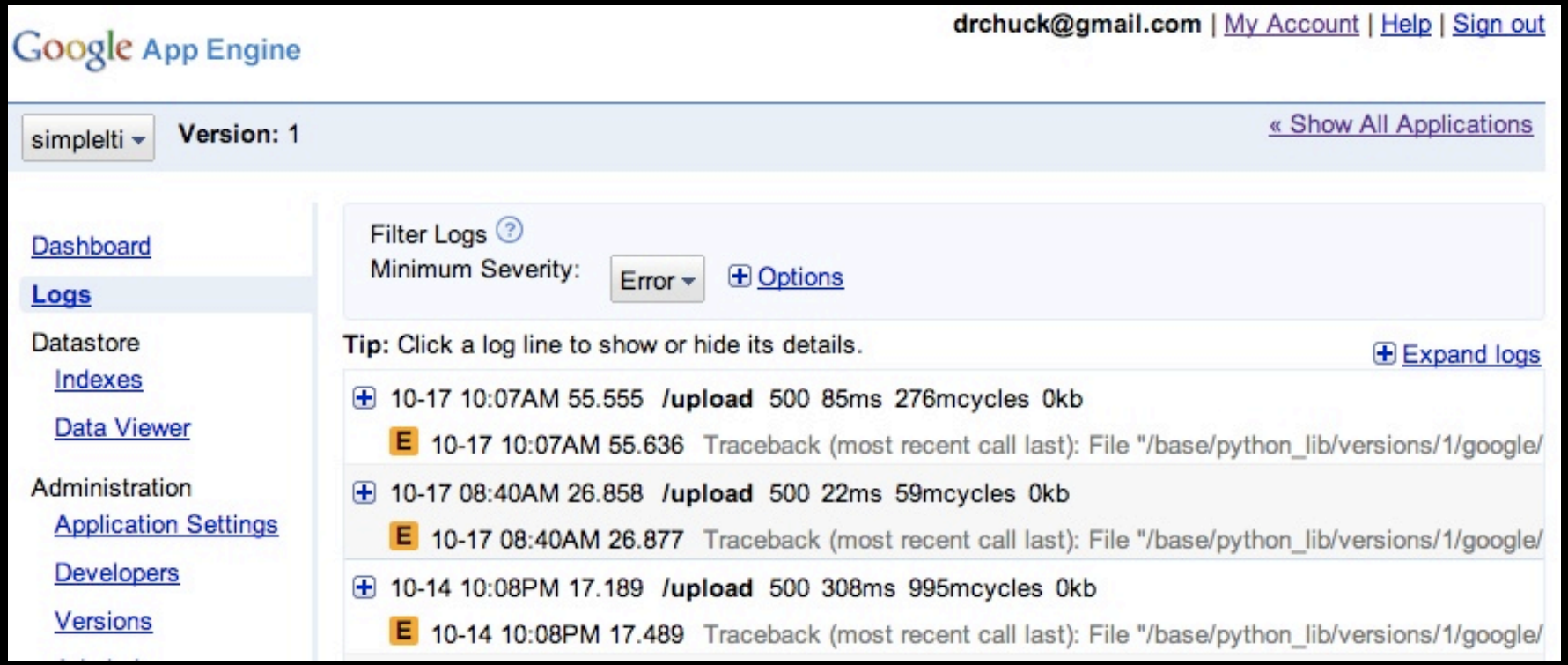
Digression: Logging

- Web Application Logging is your friend
- Your customers will never tell you when something goes wrong - they won't call you and tell you what happened
- So web applications log to a file or to a display - so you can monitor what is going on - even when someone else is using your applicaiton

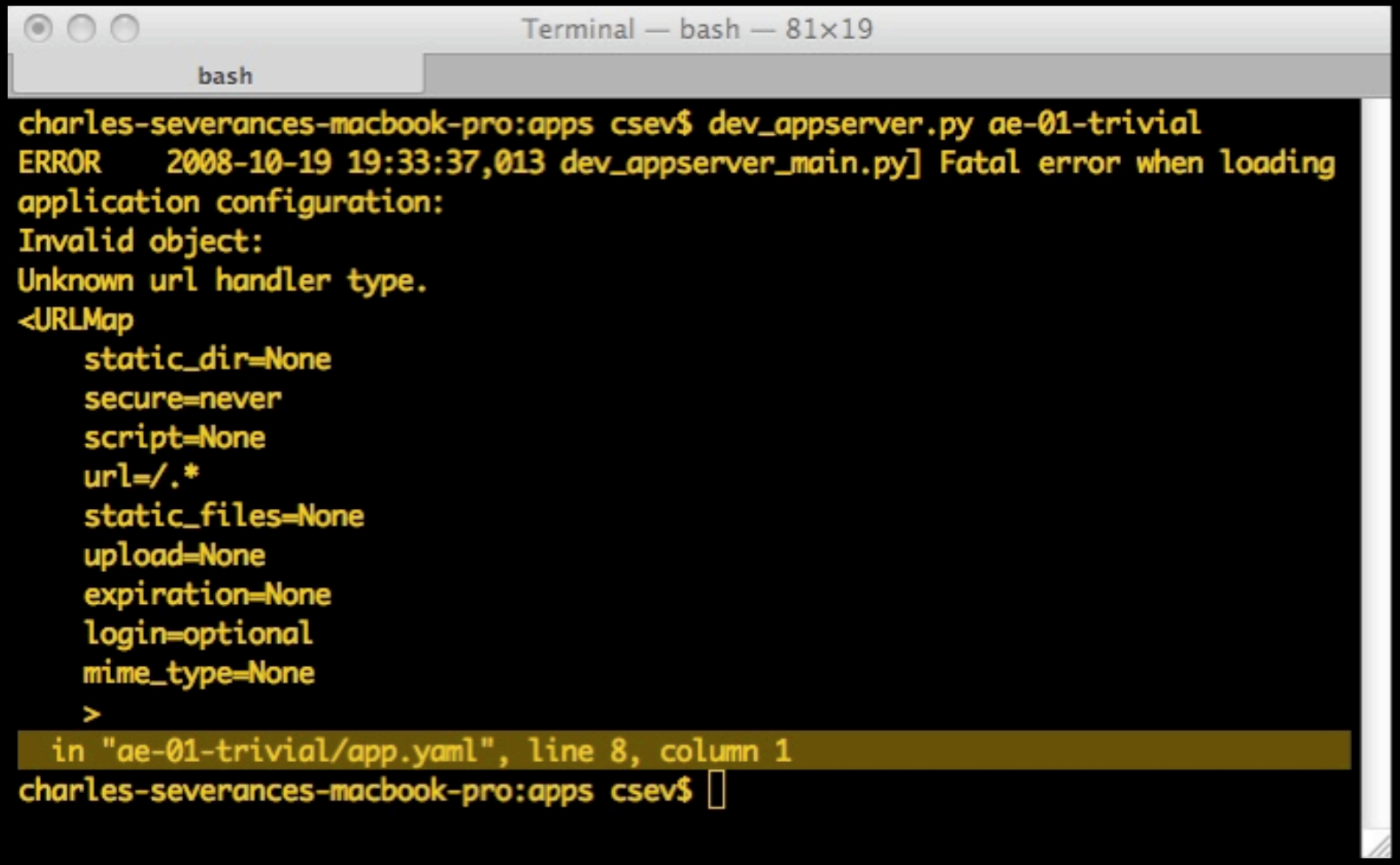
You Have Seen the Log



The log from Google



Errors in the Log



In Your Program

- The framework logs certain things on your behalf
  - Incoming GET and POST responses
  - Errors (including traceback information)
- You can add your own logging messages
  - logging.info("A Log Message")
  - FIve levels: debug, info, warning, error and critical

http://code.google.com/appengine/articles/logging.html

GET /

<html> ...

Web Appplication

Hello GET
Hello POST
...

```
class PointlessHandler(webapp.RequestHandler):

  def get(self):
    logging.info("Hello GET")

  def post(self):
    logging.info("Hello POST")
```

Back to: A Pointless Handler

```
class PointlessHandler(webapp.RequestHandler):

  def get(self):
    logging.info("Hello GET")

  def post(self):
    logging.info("Hello POST")
```

This handler, handles a GET and POST request and then does not do anything particularly useful. The post() and get() methods are the contact points between the webapp framework and our code. Our job is to prepare the response to the GET and POST requests in these methods.

The MainHandler

```
class MainHandler(webapp.RequestHandler):

  def get(self):
    logging.info("Hello GET")
    self.dumper()

  def post(self):
    logging.info("Hello POST")
    self.dumper()
```

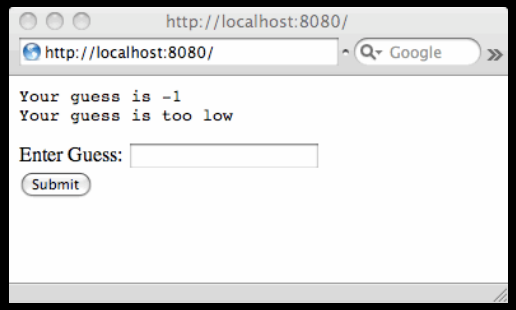
In addition to a happy little log message, the get() and post() methods both call dumper() to return a response with a form and the dumped data.

Review: Guessing CGI-Style

Web Server

HTTP Request

Browser




```
POST /
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
Content-type: application/x-www-form-urlencoded
Content-length: 8
guess=25
```

```
<form method="post" action="/">
<p>Enter Guess:
<input type="text" name="guess"/></p>
<p><input type="submit"></p>
</form>
```

index.py

```
import sys

print 'Content-Type: text/html'
print ''
print '<pre>'

# Read the form input which is a single line as follows
# guess=42
data = sys.stdin.read()
# print data
try:
   guess = int(data[data.find('=')+1:])
except:
   guess = -1
   print 'Your guess is too high'
```

```
import sys

print 'Content-Type: text/html'
print ''
print '<pre>'

# Read the form input which is a single line as follows
# guess=42
data = sys.stdin.read()
# print data
try:
   guess = int(data[data.find('=')+1:])
except:
   guess = -1
   print 'Your guess is too high'
```

```
import sys

print 'Content-Typ
print ''
print '<pre>'

# Read the form ir
# guess=42
data = sys.stdin.read()
# print data
try:
   guess = int(data[data.find('=')+1:])
except:
   guess = -1
   print 'Your guess is too high'
```

```
POST /
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
Content-type: application/x-www-form-
urlencoded
Content-length: 8
guess=25
```

```
guess=2
5
```

guess=25

```
guess = int(data[data.find('=')+1:])
```

guess=25

5

```
guess = int(data[data.find('=')+1:])
```

guess=25

5 6

```
guess = int(data[data.find('=')+1:])
```

5 6

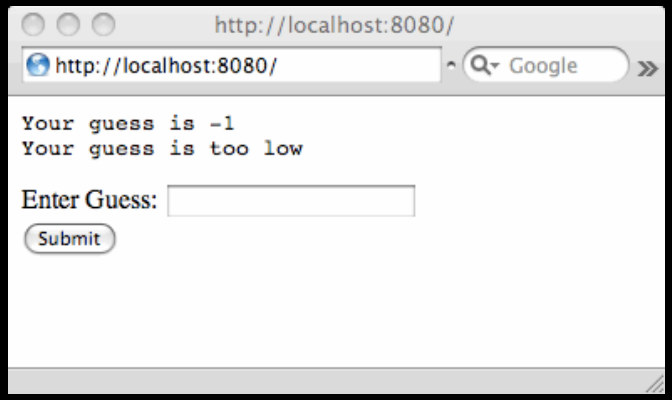
```
guess = int(data[data.find('=')+1:])
```

```
import sys

print 'Content-Type: text/html'
print ''
print '<pre>'

# Read the form input which is a single line as follows
# guess=42
data = sys.stdin.read()
# print data
try:
   guess = int(data[data.find('=')+1:])
except:
   guess = -1
   print 'Your guess is too high'
```

```
guess=25
```

```
print 'Your guess is', guess

answer = 42
if guess < answer :
   print 'Your guess is too low'
if guess == answer:
   print 'Congratulations!'
if guess > answer :
   print 'Your guess is too high'

print '</pre>'
print '''<form method="post" action="/">
<p>Enter Guess: <input type="text" name="guess"/></p>
<p><input type="submit"></p>
</form>'''
```

```
print 'Your guess is', guess

answer = 42
if guess < answer :
   print 'Your guess is too low'
if guess == answer:
   print 'Congratulations!'
if guess > answer :
   print 'Your guess is too high'

print '</pre>'
print '''<form method="post" action="/">
<p>Enter Guess: <input type="text" name="guess"/></p>
<p><input type="submit"></p>
</form>'''
```



Guess (again) as a WebApp

app.yaml

Nothing is new here

```
application: ae-03-webapp
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
  script: index.py
```

```
def main():
  application = webapp.WSGIApplication(
      [('/.*', MainHandler)],
      debug=True)
  wsgiref.handlers.CGIHandler().run(application)

if __name__ == '__main__':
  main()
```

Bottom of file

```
import logging
import wsgiref.handlers
from google.appengine.ext import webapp

class MainHandler(webapp.RequestHandler):

  formstring = '''<form method="post" action="/">
<p>Enter Guess: <input type="text" name="guess"/></p>
<p><input type="submit"></p>
</form>'''

  def get(self):
    self.response.out.write('<p>Good luck!</p>\n')
    self.response.out.write(self.formstring)
```

We Don’t Use print

- Our task is to prepare the response and give it back to the framework - so instead of just printing the output, we call
  - self.response.out.write(“Some String”)
- This lets the framework do something tricky (or Cloud-Like) with our response - if it so desires

```
def post(self):
  stguess = self.request.get('guess')
  logging.info('User guess='+stguess)
  try:
    guess = int(stguess)
  except:
    guess = -1

  answer = 42
  if guess == answer:
    msg = 'Congratulations'
  elif guess < 0 :
    msg = 'Please provide a number guess'
  elif guess < answer:
      ........
```

```
answer = 42
if guess == answer:
  msg = 'Congratulations'
elif guess < 0 :
  msg = 'Please provide a number guess'
elif guess < answer:
  msg = 'Your guess is too low'
else:
  msg = 'Your guess is too high'

self.response.out.write('<p>Guess:'+stguess+'</p>\n')
self.response.out.write('<p>'+msg+'</p>\n')
self.response.out.write(self.formstring)
```

```
def main():
  application = webapp.WSGIApplication(
      [('/.*', MainHandler)],
      debug=True)
  wsgiref.handlers.CGIHandler().run
(application)

if __name__ == '__main__':
  main()
```

Bottom of file

Summary

- We are now using the webapp framework provided by Google to handle the low-level details of the Request/ Response cycle and data formats
- We create a Handler to handle the incoming requests and then start the webapp framework to handle the requests and call our Handler as needed
- In a web application, log messages are your friend!