Lecture 10(-ish)

Web [Application] Frameworks
Minimal Python server

```python
import SocketServer
import SimpleHTTPServer

class Reply(SimpleHTTPServer.SimpleHTTPRequestHandler):
    def do_GET(self):
        # query arrives in self.path; return anything, e.g.,
        self.wfile.write("query was %s
" % self.path)

def main():
    # do initialization or whatever
    SocketServer.ForkingTCPServer('', 8080),
    Reply).serve_forever()

main()
```
Overview of server frameworks

• client-server relationship is stereotypical
  – client sends requests using information from forms
  – server parses request, calls proper function, which retrieves from database, formats response, returns it

• REST: URL filenames often used to encode requests
  …/login/name
  …/add/data_to_be_added
  …/delete/id_to_delete

• server uses URL pattern to call proper function with right arguments

• server usually provides structured & safer access to database
• server may provide templating language for generating HTML
  – e.g., replace { % foo % } with value of variable foo, etc.
• framework may automatically generate an administrative interface
• often library routines for user ids, passwords, registration, etc.
Flask: Python-based microframework

- simplest example?

```python
import flask
app = flask.Flask(__name__)
@app.route('/
@def hello():
   return 'Hello'
app.run()

$ python hello0.py

Hello
```
Sending form data

<form name=top id=top METHOD=POST
ACTION="http://localhost:5000">
<p> Name:  <input type="text" name=Name id=Name >
<p> Netid:  <input type="text" name=Netid id=Netid >
<p> Class:
<input type="radio" name=Class value="2017"> '17
<input type="radio" name=Class value="2018"> '18
<input type="radio" name=Class value="2016"> '19
<input type="radio" name=Class value="2020"> '20

<p> Courses:
<input type="checkbox" name=C126> 126
<input type="checkbox" name=C217> 217
<input type="checkbox" name=C226> 226
</ul>

<p> <input type="submit" value="Submit"> <input type=reset>
Processing form data

# survey.py

from flask import Flask, request

app = Flask(__name__)

@app.route('/', methods=['POST','GET'])
def survey():
    s = ""
    for (k,v) in request.form.iteritems():
        s = "%s %s=%s<br>" % (s, k, v)
    return 'Hello<br>' + s

app.run()
Python @ decorators

• a way to insert or modify code in functions and classes
  ```python
  @decorate
  def foo(): ...  
  ```
• compilation compiles foo, passes the object to decorate, which does something and replaces foo by the result
• used in Flask to manage URL routing

```python
@app.route('/add', methods=['POST'])
def add_entry():
    blog.insert({'title': request.form['title'],
                 'text': request.form['text']})
    return redirect(url_for('show_entries'))

@app.route('/login', methods=['GET', 'POST']) ...
@app.route('/clear', methods=['GET', 'POST']) ...
@app.route('/logout') ...
```
Django: more heavyweight Python-based framework

- by Adrian Holovaty and Jacob Kaplan-Moss (released July 2005)

- a collection of Python scripts to
  - create a new project / site
    - generates Python scripts for settings, etc.
    - configuration info stored as Python lists
  - create a new application within a project
    - generates scaffolding/framework for models, views
  - run a development web server for local testing

- generate a database or build interface to an existing database
- provide a command-line interface to application
- create an administrative interface for the database
- run automated tests
  - ...

Django Reinhart, 1910-1953
Conventional approach to building a web site

- user interface, logic, database access are all mixed together

```python
import MySQLdb
print "Content-Type: text/html"
print
print "<html><head><title>Books</title></head>"
print "<body>"
print "<h1>Books</h1>"
print "<ul>"
connection = MySQLdb.connect(user='me', passwd='x', db='my_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM books ORDER BY pub_date DESC")
for row in cursor.fetchall():
    print "<li>%s</li>" % row[0]
print "</ul>"
print "</body></html>"
connection.close()
```
Model-View-Controller (MVC) pattern

• an example of a design pattern
• model: the structure of the data
  – how data is defined and accessed
• view: the user interface
  – what it looks like on the screen
  – can have multiple views for one model
• controller: how information is moved around
  – processing events, gathering and processing data,
    generating HTML, ...
• separate model from view from processing so that when one
changes, the others need not
• used with varying fidelity in
  – Django, App Engine, Ruby on Rails, XCode Interface Builder, ...
• not always clear where to draw the lines
  – but trying to separate concerns is good
Django web framework

• write client code in HTML, CSS, Javascript, ...
  – Django template language helps separate form from content

• write server code in Python
  – some of this is generated for you

• write database access with Python library calls
  – they are translated to SQL database commands

• URLs on web page map mechanically to Python function calls
  – regular expressions specify classes of URLs
  – URL received by server is matched against regular expressions
  – if a match is found, that identifies function to be called and arguments to be provided to the function
Example database linkage

```python
dATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/Users/bwk/django/sql3.db', ...
    }
}

from django.db import models
class Post(models.Model):
    title = models.TextField(5)
    text = models.TextField()

BEGIN;
CREATE TABLE "blog_post" (  
    "id" integer NOT NULL PRIMARY KEY,
    "title" text NOT NULL,
    "text" text NOT NULL
)
;
generated by Django
```

in settings.py

in models.py
URL patterns

- regular expressions used to recognize parameters and pass them to Python functions
- provides linkage between web page and what functions are called for semantic actions

```python
urlpatterns = patterns('',
    (r'^time/$', current_datetime),
    (r'^time/plus/([\d{1,2}])/$', hours_ahead),
)
```

- a reference to web page `.../time/` calls the function
  ```python
current_datetime()
```
- tagged regular expressions for parameters: url `.../time/plus/12` calls the function
  ```python
hours_ahead(12)
```
Templates for generating HTML

- try to separate page design from code that generates it
- Django has a specialized language for including HTML within code
  - loosely analogous to PHP mechanism

```html
# latest_posts.html (the template)

<html><head><title>Latest Posts</title></head>
<body>
<h1>Posts</h1>
<ul>
{% for post in post_list %}
  <li>{{ post.title }} {{ post.text }}</li>
{% endfor %}
</ul>
</body></html>
```
Administrative interface

• most systems need a way to modify the database even if initially created from bulk data
  – add / remove users, set passwords, ...
  – add / remove records
  – fix contents of records
  – ...

• often requires special code

• Django generates an administrative interface automatically
  – loosely equivalent to MyPhpAdmin
Alternatives…

• Ruby on Rails

• Google App Engine

• Node + Express

• Google Web Toolkit

• and lots of others
Node.js server

```javascript
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
```

- Express framework for Node
  - analogous to Flask
Assessment of Web Frameworks

• **advantages**
  – takes care of repetitive parts
    more efficient in programmer time
  – automatically generated code is likely to be more reliable, have more
    uniformity of structure
  – "DRY" (don't repeat yourself) is encouraged
  – "single point of truth"
    information is in only one place so it's easier to change things
  – ...

• **potential negatives**
  – automatically generated code
    can be hard to figure out what's going on
    can be hard to change if you don't want to do it their way
  – systems are large and can be slow
  – ...

• **read Joel Spolsky's "Why I hate frameworks"**
  http://discuss.joelonsoftware.com/default.asp?joel.3.219431.12