Framework user to author in four hours

Fredrik Håård
fredrik.haard@softhouse.se

Softhouse Consulting
www.softhouse.se

http://blaag.haard.se/framework.tar.gz
whoami
Prerequisites

• cPython 3.2+
• Exercises at http://blaag.haard.se/framework.tar.gz
Rationale (aka "why not just use Django/Flask/CherryPy/…")

- Large frameworks are written for everyone
- Frameworks are written by people
Session plan

- We (you) will write a web application framework
- You implement exercises however you want
- Example code for each part is available
A simple WSGI app

- wsgiref
- Includes a validator to make sure your WSGI app / framework is compliant to the specification (PEP-3333)

```python
def application(environ, start_response):
    start_response('200 OK',
                   [('Content-type', 'text/plain')])
    return [b"Hello World"]

httpd = make_server('', 8000, application)
httpd.serve_forever()
```
One: write a basic wsgi app

• Use the validator example from wsgiref
• Print the environ dict to see what data is available
Routing requests

- Choice of request routing
- regex, dict/tree, package/module tree, ???

```python
def resource(environ):
    return b"Hello world"

def application(environ, start_response):
    ...
    if re.match('^/resource/.*', environ['PATH_INFO']):
        yield resource(environ)
```
Two: Routing

- Create routing logic for request, for example using regex and environ['PATH_INFO']
- Make separate handlers for /, /resource, and /resource/nested
Detecting and loading plugins

- Find plugin implementations
- Compile/import code
- `importlib` is useful
- `imp` works, but is the old way
for <python file> in <path>:
    module = importlib.find_loader(
        <name>, [<path>]).load_module()
    module.do_interesting_stuff()
Three: Plugin loading

- Locate and load plugins
- Example plugin in ‘plugins/plug.py’
- How a plugin/webapp is loaded will depend on your routing mechanic
Live access to running process

• Not a trivial problem
• `code` library has working interpreters
• `stdout` / `stderr` redirection
• `print` / `displayhook` replacement
• Roll your own interpreter
REPL example

def display(out):
    def dsp(x):
        if x: out.write((str(x)+'
').encode())
    return dsp

def handle(self):
    sys.displayhook = display(self.wfile)
    interpreter = code.InteractiveConsole()
    self.wfile.write(b'>>> ')
    while True:
        data = self.rfile.readline()[:-1]
        if interpreter.push(data.decode()):
            self.wfile.write('... '.encode())
        else:
            self.wfile.write('>>> '.encode())
Four: REPL access to server process

- Connect a Socketserver to an interpreter
- Either redirect stdout/err
- or overload displayhook, print, take care of exceptions...

- socketserver_scaffold.py
  basic TCP echo server
Reloading code on the fly

- Polling, filesystem events, or check on access
- How to handle broken code?
- Restart server or reload specific parts?
Reloading modified code

```python
import my_module
my_module.load_time = time.time()
...
if (my_module.load_time < os.stat(my_module.__file__).st_mtime):
    my_module = imp.reload(my_module)
```
Five: hot code swapping

- Detect source changes in plugins, and reload code
- `use imp.reload` to reload changed modules
Pipes and filters

- Adding authentication, data transformation (JSON/HTML/???) etc requires that we can call several handlers
- A very simple way is to just have a list of handlers and execute all that matches the request
Six: Pipes and filters

- Add support for multiple handlers for a single request
- Add an authentication plugin that requires auth data in the query string
- Add a plugin that serves static content from /static
Creating an internal DSL

• Example: Django
  
  ```python
  class Poll(models.Model):
      question = models.CharField(max_length=200)
      pub_date = models.DateTimeField('date')
  ```

• Using Python syntax to simplify specifying domain-specific stuff

• Can make applications faster to write/modify, with less chances of breakage

• Can enforce style / limitations
DSL Example using functions (page.py)

```python
page = page(
    header(title='This is a page'),
    body(
        heading("This is a heading"),
        paragraph("This is a paragraph with some interesting
        content, written twice. This is a paragraph with some interesting
        content, written twice. "")
    ),
    form(method='get', action='/', title='A form',
    fields=(
        dict(
            name='user',
            fieldtype='text',
            value=''
        ),
        dict(
            name='send',
            value='Ok',
            fieldtype='submit'
        )
    ))
)
```
Seven: DSL

- Make an app/plugin that renders page.py properly
- Implement `page_renderer.py`
- Hook it up to a handler