# Python programming — Web serving

Finn Årup Nielsen

Department of Informatics and Mathematical Modelling
Technical University of Denmark

September 30, 2013

# Overview

Methods

Model-view-controller

Web-frameworks

Cloud-based services

CGI, Cherrypy

Templating

Profiling

Google App Engine

# Methods

## CGI

Simple. "Common Gateway Interface". Simple parsing of query string and make its parts available as Python variables.
`http://docs.python.org/library/cgi.html`

## Frameworks

Cherrypy: Light-weight. Includes your own web-server in Python.

Django: Major Python framework

Others: Flask ("microframework"), web2py ("web-based"), Pylons, Zope (Plone), ...

# Model-view-controller (MVC)

**Model** may be the object-relational mapper for storing the data, e.g., `sqlobject`, `sqlalchemy`.

**View**. Web templating system where the HTML is defined, e.g., `Cheetah.Template`, `tornado.template`, . . . Called the "template" in Django.

**Controller** translates URLs into actions using the "model" to set and get data and render the result using the view, e.g, `cherrypy` itself. Called the "view" in Django.

# Web frameworks

| System | Simple | CherryPy | Django | Tornado |
|---|---|---|---|---|
| Web server | Apache | CherryPy | Apache | Tornado |
| Model | e.g. `pysqlite` | e.g. `sqlobject` | `django.db` | `tornado.database` |
| View | `print` | e.g. `Cheetah` | `django.template` | `tornado.template` |
| Controller | (`cgi`) | `cherrypy` | `django.view` | `tornado.web` |

Table 1: Web frameworks

"LAMP": Linux, Apache, MySQL, Python (or PHP/Perl)

Tighter integration with Apache: `mod_wsgi` (and `mod_python`).

# Where to run Python?

| System | kr/month | Comments |
|---|---|---|
| DTU G-bar | 0 | No root access, e.g., to setup MySQL. Missing Python packages |
| You own computer | 0 | May be difficult to get on the Internet. May not be able to run all the time |
| one.com | 11.25 | Not possible with Python. Only PHP. |
| linode.com | 110.00 | Virtual linux server with root access |
| Heroku | 210/0 | "Web dyno". Extra cost for "workers" and database |
| Google App Engine | 0(?) | Python 2.5. Additional cost above quota. GQL. Some Python modules may be missing |
| PythonAnywhere | 0–60 | Python hosting |
| ... | ? | |

# cgi

http://docs.python.org/library/cgi.html

```
import cgi              # Not from cgi import *

import cgitb            # This module displays error message if something
cgitb.enable()          # goes wrong. Good for debugging.

form = cgi.FieldStorage()          # 'form' is a dictionary-like object
query_string = form.getvalue('q')
```

`query_string` now contains the content from the "q" parameter in the URL, e.g.:

```
http://yourhost.com/yourscript?q=get+rich+quick
```

```
query_string = "get rich quick"
```

# Bad way

A cgi web script called `badescape`:

```
#!/usr/bin/env python
import cgi
name = cgi.FieldStorage().getvalue('name', '')
print("""Content-Type: text/html\n
<html><body><form><input name="name" value="%s"> Your name is %s
</form></body></html>""" % (name, name))   # No escaping on 'name'!!!
```

It works ok with normal input:

```
http://localhost/cgi-bin/badescape?name=Finn
```

But an evil user may also call it with embedded javascript (on one line):

```
http://localhost/cgi-bin/badescape?name=Finn"><script+type%3D"text%2F
javascript">+window.open("http%3A%2F%2Fwww.dtu.dk%2F"%2C+"Buy+Viagra")
<%2Fscript>
```

# Better way

```
#!/usr/bin/env python
import cgi, urllib
name = cgi.FieldStorage().getvalue('name', '')
print("""Content-Type: text/html\n
<html><body><form><input name="name" value="%s"> Your name is %s
</form></body></html>""" % (urllib.quote(name), cgi.escape(name)))
```

The escaping is used to make sure that "<", ">" and "&" are escaped to "<", ">" and "&" or the percentage encoding in the URL.

`cgi.escape(name, True)` can escape to """. Use `quoteattr()` from the `xml.sax.saxutils` module to escape both single and double quotes.

Note also the problem of Unicode/UTF-8 encoding: `name.encode("utf-8"))`

# cgi: slightly larger example

```
#!/usr/bin/env python
import cgi, cgitb, re, simplejson
cgitb.enable()
form = cgi.FieldStorage()
q = form.getvalue('q', '')
lang = form.getvalue('lang', 'en')
if lang not in ['da', 'en']: lang = en                    # Sanitize 'lang'
page = re.findall(r'\d+', form.getvalue('page', '1')) # 'page' an integer
if page:
    page = int(page[0])
else:
    page = 1
checked = {'en': 'checked', 'da': ''}
if lang == 'da':
    checked = {'da': 'checked', 'en': ''}
```

```
print("""Content-Type: text/html\n
<html>
  <body>
    <form>
      <input name="q" value="%s"> <input name="page" value="%d">
      <input type="radio" name="lang" value="en" %s>English
      <input type="radio" name="lang" value="da" %s>Dansk
      <input type="submit" value="Search">
    </form>
  </body>
</html>""" % (cgi.escape(q, True), page, checked['en'], checked['da']))
```

# Web frameworks

# Cherrypy

Cherrypy: A simple framework to get you started

# Cherrypy Hello world example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import cherrypy, sys
reload(sys)
sys.setdefaultencoding("utf-8")

class HelloWorld(object):
  def index(self):
    return u"""<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01""" + \
           """Transitional//EN">\n\n""" + \
           "<html><body><h1>Hello, Wørld</h1></body></html>"
  index.exposed = True


cherrypy.root = HelloWorld()
cherrypy.server.start()
```

# Cherrypy hello world result

Save the code in a file and execute it, then the web server starts on default localhost port 8080 (here with "World" rather than "Wørld")

# Cherrypy with an argument

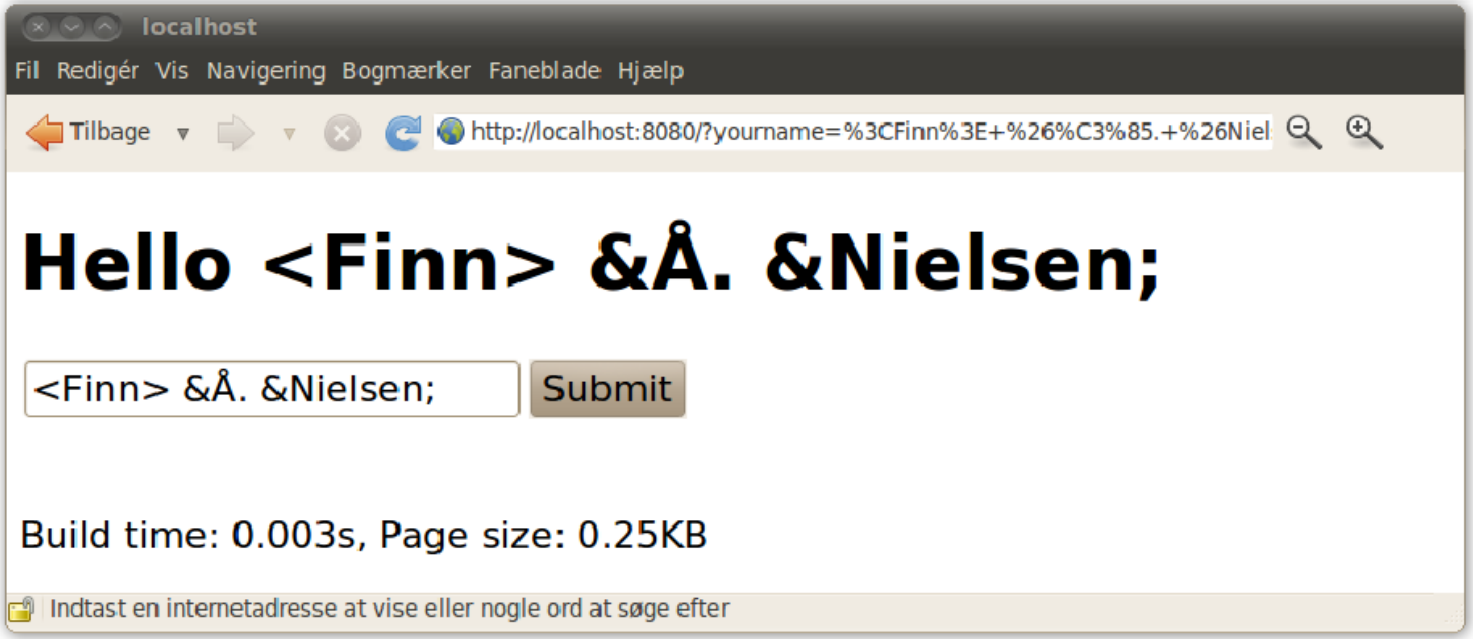
```
#!/usr/bin/env python
import cgi, cherrypy, sys
reload(sys)
sys.setdefaultencoding("utf-8")

class HelloYou(object):
  def index(self, yourname=""):
    return """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01""" + \
           """Transitional//EN">\n\n""" + \
           """<html><body><h1>Hello %s</h1>
              <form>
                <input name="yourname"><input type="submit">
              </form>
              </body></html>""" % (cgi.escape(yourname))
  index.exposed = True
```

# Cherrypy with an argument: The result

Configuring and starting the webserver with:

```
cherrypy.root = HelloWorld()
cherrypy.server.start()
```



# Cherrypy with multiple pages

```
#!/usr/bin/env python
import cherrypy

class MultiplePages(object):
  def header(self):
    return "<html><body>"
  def footer(self):
    return "</body></html>"
  def index(self):
    return self.header() + '<a href="s">s</a>' + self.footer()
  index.exposed = True
  def s(self):                  # Method for another page
    return self.header() + '<a href="..">..</a>' + self.footer()
  s.exposed = True

cherrypy.root = MultiplePages()
cherrypy.server.start()
```

# Templating

Cheetah

Jinja2 — Used in Google App Engine

Django — The Django web framework has its own templating module.

Mako — default in Pylons and Pyramid web frame works. Used on the python.org website

`tornado.template` (for Tornado)

. . . and others

# Jinja2 example

Jinja example:

```
>>> from jinja2 import Template
>>> tmpl = Template(u"""<html><body><h1>{{ name|escape }}</h1>
                       </body></html>""")
>>> tmpl.render(name = u"Finn <Årup> Nielsen")
u'<html><body><h1>Finn &lt;\xc5rup&gt; Nielsen</h1></body></html>'
```

Note here the escaping of the less than and bigger than signs.

# Benchmarking example

Cherrypy probably not suitable to large-scale deployment with its own server.

Benchmarking with the “ab” program

```
$ ab -c 10 -n 10000 -k localhost.localdomain:8080/
```

For speed consider `mod_wsgi` or, e.g., Tornado.

Tornado profiling:

```
$ wget https://raw.github.com/facebook/tornado/master/demos/helloworld/helloworld.py
$ python helloworld.py
$ ab -c 100 -n 1000 -k localhost.localdomain:8888/ | grep "Time taken for tests:"
Time taken for tests:   0.707 seconds
```

CherryPy profiling:

```
$ wget https://raw.github.com/gist/3819040/c05d432a0d4e261b37ef6a55b2a8f7d4f4584c76/helloc
$ python hellocherry.py
$ ab -c 100 -n 1000 -k localhost.localdomain:8080/ | grep "Time taken for tests:"
Time taken for tests:   21.649 seconds
```

# Getting started with Google App Engine

Download Google App Engine SDK for your architecture

Install, e.g., `unzip google_appengine_1.8.1.zip` and
`PATH=${PATH}:/opt/google_appengine`

Register an application at https://appengine.google.com You need a Google account.

Make a directory with, e.g., helloworld-like fnielsentest.py and app.yaml

Start local version `dev_appserver.py fnielsentest/` and check out http://localhost:8080

Upload to cloud, e.g., `appcfg.py --oauth2 update fnielsentest/`

Check out http://fnielsentest.appspot.com/

# More information

Cherrypy tutorial:

```
>>> import cherrypy.tutorial
>>> cherrypy.tutorial.__file__
'/usr/lib/python2.7/dist-packages/cherrypy/tutorial/__init__.pyc'
```

Especially `bonus-sqlobject.py` shows a good example.

For the Pylon web framework, see, e.g., Introduction to Pylons chapter in *The Definitive Guide To Jython*.

"Programming Google App Engine".