

Simplistix

**Python Web Framework
Templating Systems
Compared and Contrasted**

Chris Withers

Who am I?

- Chris Withers
- Independent Zope and Python Consultant
- Using Zope and Python since 1999

- 90% of what I do is web-based

- I like things as simple as possible...
...but no simpler

What am I going to cover?

- focus on templating styles
 - there are enough of them!
- gloss over publishing process
 - hand-waving expert
- ignore all but minimal aspects
 - because that's all there should be!

A long time ago...

...in a galaxy far far away:

```
class Folder:

    def __init__(self):
        self.pages = []

    def addPage(self,p):
        self.pages.append(p)
        p.folder = self

    def getPages(self):
        return tuple(self.pages)
```

```
class Page:

    status,folder = 'private',None

    def __init__(self,title,content):
        self.title = title
        self.content = content

    def publish(self):
        self.status = 'published'

    def getURL(self):
```

```
folder = Folder()
page1 = Page('This is my page!','This is the content of the page.')
page2 = Page('Page 2','This is the content of page 2.')
page3 = Page('Page 3','This is the content of page 3.')
folder.addPage(page1);folder.addPage(page2);folder.addPage(page3)
```

Then came the web!

```
<html>
  <head>
    <title>This is my page!</title>
  </head>
  <body>
    <h1>This is my page!</h1>
    <div id="body">
      <div id="content">
        This is the content of the page.
      </div>
      <div id="tagline">
        This content is <b>published</b>.
      </div>
    </div>
    <ul id="leftnav">
      <li><a href="page1.html" class="selected">This is my page!</a></li>
      <li><a href="page2.html">Page 2</a></li>
      <li><a href="page3.html">Page 3</a></li>
    </ul>
  </body>
</html>
```

Oh crap, we need to generate that stuff!

```
print '<html>'
print '  <head>'
print '    <title>%s</title>' % page.title
print '  </head>'
print '  <body>'
print '    <h1>%s</h1>' % page.title
print '    <div id="body">'
print '      <div id="content">%s</div>' % page.content
print '      <div id="tagline">'
print '        This content is <b>%s</b>.' % page.status
print '      </div>'
print '    </div>'
print '    <ul id="leftnav">'
for p in page.folder.getPages():
    print '      <li><a href="%s" class="%s">%s</a></li>' % (
        p.getURL(),
        p.is_page and 'selected' or 'normal',
        p.title
    )
print '  </ul>'
print '  </body>'
print '</html>'
```

But we're not using CGI!

- Python 2.4 string templating

```
from string import Template

template = Template('''<html>
  <head>
    <title>$title</title>
  </head>
  <body>
    <h1>$title</h1>
    <div id="body">
      <div id="content">$content</div>
      <div id="tagline">
        This content is <b>$status</b>.
      </div>
    </div>
    <ul id="leftnav">
      $leftnav
    </ul>
  </body>
</html>''')
```

But we're not using CGI!

```
def page_view(page):  
  
    nav = []  
    for p in page.folder.getPages():  
        nav.append('    <li><a href="%s" class="%s">%s</a></li>' % (  
            p.getURL(),  
            p is page and 'selected' or 'normal',  
            p.title  
        ))  
  
    return template.substitute(  
        'title':page.title,  
        'content':page.content,  
        'status':page.status,  
        'leftnav':'\n'.join(nav),  
    )
```

- What about things other than pages?

Quixote - PTL

- <http://quixote.ca/learn/1>
- almost python classes
- components instantiated by framework
- designed for Quixote's model

Quixote - PTL

- The site template

```
import quixote
quixote.enable_ptl()

def header [html] (title, stylesheet):
    '<html>'
    ' <head>'
    '   <title>%s</title>' % title
    ' </head>'
    ' <body>'
    ' <h1>%s</h1>' % title
    ' <div id="body">'

def footer [html] (page):
    ' </div>'
    ' <ul id="leftnav">'
    for p in page.folder.getPages():
        klass = p is page and 'selected' or 'normal'
        '<li><a href="%s" class="%s">%s</a></li>' % (p.getURL(), klass, p.title)
    ' </ul>'
    ' </body>'
    '</html>'
```

Quixote - PTL

- The page

```
from template import header, footer

class PageUI:

    _q_exports = ['_q_index']

    def __init__(self, page):
        self.page = page

    def _q_index [html] (self, request):
        header()
        '<div id="content">'
        self.page.content
        '</div>'
        '<div id="tagline">'
        'This content is <b>%s</b>.' % self.page.status
        '</div>'
        footer()
```

Quixote - PTL

- Positives
 - allows template re-use
 - fits with Quixote framework
 - all in python, but not quite python
- Negatives
 - all in python, but not quite python
 - broken html possible
 - what would an html-only person think?

Web Monkeys

- We are programmers
- We don't *really* like HTML
- We *really* don't like CSS

- Give the problem to someone who cares...

- ...but in way that we can still do our job

Zope - DTML

- http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/DTML.stx
- Actually a generic scripting language
- Highly tied to Acquisition
- Single layered namespace
- “Tell the web monkeys to leave *<dtml-anything>* alone”

Zope - DTML

- standard_html_header

```
<html>
  <head>
    <title><dtml-var title></title>
  </head>
  <body>
    <h1><dtml-var header></h1>
    <div id="body">
```

- standard_html_footer

```
</div>
<ul id="leftnav">
  <dtml-in getPages>
    <li><a href="<dtml-var getURL>"
      <dtml-if "_getitem('sequence-item') is this()">
        class="selected"</dtml-if>>
      <dtml-var title></a></li>
  </dtml-in>
</ul></body></html>
```

Zope - DTML

- The page

```
<dtml-var standard_html_header>
  <div id="content">
    <dtml-var content>
  </div>
  <div id="tagline">
    This content is <b><dtml-var status></b>.
  </div>
<dtml-var standard_html_footer>
```

Zope - DTML

- Positives
 - can often DWIM
 - simple templates
 - not just XML/HTML
- Negatives
 - Acquisition
 - One big namespace
 - Funky special variable names
 - What about HTML editors?

Zope - ZPT

- http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/ZPT.stx
- Zope templating mk II
- designed for visual html editors
- separation of logic and presentation

Zope - ZPT

- main_template.pt

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      metal:define-macro="page">
  <head>
    <title tal:content="context/title">This is my page!</title>
  </head>
  <body>
    <h1 tal:content="context/title">This is my page!</h1>
    <div id="body"
      <metal:x define-slot="body">
        Body content goes here!
      </metal:x>
    </div>
    <ul id="leftnav">
      <li tal:repeat="page context/folder/getPages">
        <a href="page1.html" class="selected"
          tal:attributes="href page/getURL;
                        class python:context is page;"
          tal:content="page/title">This is my page!</a></li>
    </ul>
  </body>
</html>
```

Zope - ZPT

- The page

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      metal:use-macro="context/main_template.pt/macros/page">
  <body>
    <metal:x fill-slot="body">
      <div id="content"
          tal:content="context/body">
        This is the content of the page.
      </div>
      <div id="tagline">
        This content is <b tal:content="context/status">published</b>.
      </div>
    </metal:x>
  </body>
</html>
```

Zope - ZPT

- Positives
 - clean namespaces
 - harder to introduce business logic
 - source is valid xml/html
- Negatives
 - tied to generating xml/html
 - 2 or 3 new languages
 - doesn't really work with visual editors
 - macros are confusing
 - very tied to Zope

Cheetah

- <http://www.cheetahtemplate.org/>
- similar to Django Templates
- not tied to one particular framework
- not limited to xml
- built in section caching support

Cheetah

- template

```
<html>
  <head>
    <title>${page.title}</title>
  </head>
  <body>
    <h1>${page.title}</h1>
    <div id="body">
      #block content
      This is the body of this template
    #end block
  </div>
  <ul id="leftnav">
    #for $p in $page.folder.getPages()
    <li><a href="${p.getURL()}"
      #if $p is page
      class="selected"
      #end if>${p.title}</a></li>
    #end for
  </ul>
</body>
</html>
```

Cheetah

- The page

```
#from Template import Template
#extends Template
#def body
    <div id="content">
        $page.content
    </div>
    <div id="tagline">
        This content is <b>$page.status</b>.
    </div>
#end def
```

Cheetah

- Positives
 - slightly more explicit than Django Templates
 - more python in substitutions
 - caching
- Negatives
 - too many fiddly extra bits
 - it's another language
 - we're back to not helping the web monkeys

Nevow

- <http://divmod.org/projects/nevow>
- Really aimed at twisted.nevow
- Attribute language templates
- DOM manipulation in methods
- Tied together by classes with special method names

Nevow

- template

```
<html xmlns:nevow="http://nevow.com/ns/nevow/0.1">
  <head>
    <title nevow:render="title">This is my page!</title>
  </head>
  <body>
    <h1 nevow:render="title">This is my page!</h1>
    <div id="body"
      nevow:macro="content">
      This is the body of this template
    </div>
    <ul id="leftnav"
      nevow:data="thepages" nevow:render="sequence">
      <li nevow:pattern="item" nevow:render="row">
        <a href="page1.html" class="selected">
          <nevow:attr name="href"><nevow:slot name="url"/></nevow:attr>
          <nevow:attr name="class"><nevow:slot name="class"/></nevow:attr>
          <nevow:slot name="title" />
        </a>
      </li>
    </ul>
  </body>
</html>
```

Nevow

- template python

```
from nevow import rend
from nevow import loaders

class BasePage(rend.Page):

    docFactory = loaders.htmlfile(template='site.html')

    def data_thepages(self, context, data):
        return self.folder.getPages()

    def render_row(self, context, page):
        context.tag.fillSlots('url', page.getURL())
        if page is self.page:
            klass = 'selected'
        else:
            klass = 'normal'
        context.tag.fillSlots('class', klass)
        context.tag.fillSlots('title', page.title)
        return context.tag
```

Nevow

- page

```
<html xmlns:nevow="http://nevow.com/ns/nevow/0.1">
  <body>
    <div id="body" nevow:pattern="body">
      <div id="content"
        nevow:render="content">
        This is the content of the page.
      </div>
      <div id="tagline">
        This content is <b nevow:render="status">published</b>.
      </div>
    </div>
  </body>
</html>
```

Nevow

- page python

```
from nevow import rend
from nevow import loaders

from template import BasePage

class Page(BasePage):

    def macro_content(self, context):
        return loaders.htmlfile(template='page.html', pattern='content')

    def render_content:
        return self.page.content

    def render_status:
        return self.page.status
```

Nevow

- Positives
 - clean separation of data, logic and presentation
 - does help the web monkeys
- Negatives
 - very verbose
 - like most things twisted
 - weird
 - not well documented
 - too much magic
 - nevow:attr - argh!

Many More...

- Myghty
 - `html::mason` for python
- Preppy
 - can handle big docs
- XSLT
 - not just python
- The “melds”
 - PyMeld, meld2, meld3
- Others
 - CherryTemplate, Formencode.htmlgen

CherryPy – Kid

- <http://kid.lesscode.org/>
- has an attribute language
 - like ZPT
- has pre-processor-like statements
 - like Django/Cheetah/PHP
- Has a substitution method
 - like string.Template
- Mash everything together!
 - ZPT, XSLT, PHP, Cheetah

Types of Templating

- Preprocessor
 - DTML
 - Django
 - Cheetah
 - Kid
- Attribute Languages
 - ZPT
 - Nevow
 - Kid
- Class-based
 - PTL
 - Nevow
- DOM-based
 - PyMeld
 - meld2
 - meld3

So what do we actually do?

```
<html>
  <head>
    <title>This is my page!</title>
  </head>
  <body>
    <h1>This is my page!</h1>
    <div id="body">
      <div id="content">
        This is the content of the page.
      </div>
      <div id="tagline">
        This content is <b>published</b>.
      </div>
    </div>
    <ul id="leftnav">
      <li><a href="page1.html" class="selected">This is my page!</a></li>
      <li><a href="page2.html">Page 2</a></li>
      <li><a href="page3.html">Page 3</a></li>
    </ul>
  </body>
</html>
```

So what do we actually do?

- replace
 - attributes
 - values
 - tags

So what do we actually do?

- repeat
 - tags
 - usually followed by a replace

So what do we actually do?

- remove
 - tags
 - attributes
 - values
 - whole nodes

Anything else?

- How do you get hold of the thing you want to work with?

So what if we just did that?

- no new languages
- work with raw html
 - don't change it from html
- as simple as possible
 - but no simpler
- (everything in one file)

Twiddler

- finding things...
- `n = t.getById('something')`
- `n = t.getName('something')`

Twiddler

- replace things...
- `n.replace(value,tag,**attributes)`
 - supplied argument is used
 - all arguments are optional
 - True means “leave as is”
 - False means “remove it”
- value can be another node

Twiddler

- repeat things...
- `n.repeat(value,tag,**attributes)`
 - return the newly inserted node
 - takes the same parameters as `replace`
 - for convenience

Twiddler

- remove things...
- `n.remove()`
 - remove the node from the current twiddler

Twiddler

- what if I want to keep it all in one place?
- code blocks
 - one per template
 - executed on render
 - executed when explicitly requested

Twiddler

- what about macros?
- `t.clone()`
 - will be cheap
 - allows partial rendering

Back to the original example...

- template

```
<html>
<!--twiddler
page = options['page']
t.getById('title').replace(page.title)
t.getById('h1_title').replace(page.title)
i = t.getByName('nav_item')
for p in page.folder.getPages():
    n = i.repeat(name=False)
    e = n.getByName('nav_link')
    e.replace(p.title,
              href=p.getURL(),
              class_=p.is_page,
              name=False)
-->
<head>
  <title name="title">This is my page!</title>
</head>
<body>
<h1 name="h1_title">This is my page!</h1>
<div id="body">This is the body of this template</div>
<ul id="leftnav">
  <li name="nav_item"><a name="nav_link" href="page1.html" class="selected">This is my
page!</a></li>
</ul>
</body>
</html>
```

Twiddler

- page

```
<html>
<!--twiddler
page = options['page']
template = options['template'].clone()
template.execute(page=page)
template.getById('body').replace(t.getById('body'))
t = template
t.getById('content').replace(page.content)
t.getName('status').replace(page.status)
-->
<body>
<div id="body">
  <div id="content">
    This is the content of the page.
  </div>
  <div id="tagline">
    This content is <b name="status">published</b>.
  </div>
</div>
</body>
</html>
```

Something cute...

- Repeating column-based table data

```
>>> t = Twiddler('''
... <html>
...   <body>
...     <table>
...       <tr>
...         <th>First Name</th>
...         <td name="firstname">John</td>
...       </tr>
...       <tr>
...         <th>Last Name</th>
...         <td name="lastname">Doe</td>
...       </tr>
...     </table>
...   </body>
... </html>
... ''')
>>> firstname = t.getBy_name('firstname')
>>> lastname = t.getBy_name('lastname')
>>> for person in people:
...     firstname.repeat(person.firstname)
...     lastname.repeat(person.surname)
```

Twiddler

- Positives
 - works with real html
 - no new languages
 - simple as possible
- Negatives
 - verbose
 - not battle-proven
 - slow?

How do I know I got it right?

- meld3
- never saw it while Twiddler was being developed
- scarily similar!

Meld3

- uses 'meld:id' attribute
- clone()
- findmeld(name)
- repeat(childname)
- replace(text,structure=False)
- content(text,structure=False)
- attributes(**kw)

But the web isn't just HTML...

- mail
- CSS
- “no xml”
- pipeline:
 - input
 - dom
 - output

```
To: $to
From: webmaster@example.com
Subject: Order No $order_do

Dear $to,

The following order is on it's way:

<items>$sku $description $quantity $code
</items>

Sincerely,

ExampleCo Order Team
```

What about?

- i18n
- html quoting
- stx
- filters
 - just for replace?
 - default filters

```
from i18n import translate
from filters import html_quote, lower

x.replace(
    someText,
    filters=(translate, html_quote)
)

y.replace(
    id='SomeThing',
    filters_='x,y,z',
    filters=(lower,)
)
```

Thankyou!

- Chris Withers
- chris@simplistix.co.uk
- <http://www.simplistix.co.uk/software/python/twiddler>
- <http://www.simplistix.co.uk/presentations>