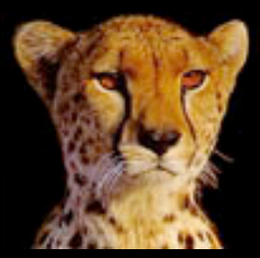

CHEETAH

The Python-Powered
Template Engine

TriZPUG :: September 24, 2009 :: Chris Calloway

What is Cheetah?

- A Template Engine

What's a Template Engine?

- Takes text as input
- Substitutes some of the input text with results of (in this case, Python) expressions
- Creates output text

What's this used for?

- Code Generation!
  - HTML
  - XML
  - SQL
  - Postscript
  - Email
  - Python
  - Java
  - PHP
  - Whatever
- Tight Integration with Webware (Python Server Pages and Servlets)

How to Cheetah

- `bin/easy_install Cheetah`

Install the standard way

How to Cheetah

- `bin/cheetah help`

Run new executable from command shell

How to Cheetah

- `from Cheetah import Template`

Use in your Python scripts

Template Definitions

- A string
  - Text (stuff that doesn't get replaced)
  - Placeholders (expressions to replace)
  - Directives (commands)
- Stored in .tmpl files

Filling a Template: Part I

```
>>> from Cheetah import Template
>>> templateDef = """
... <HTML><HEAD><TITLE>$title</TITLE></HEAD>
... <BODY>$contents</BODY></HTML>"""
>>> nameSpace = {'title': 'Hello World Example',
...              'contents': 'Hello, World!'}
>>> t = Template.Template(templateDef,
...                       searchList=[nameSpace])
>>> print t

<HTML><HEAD><TITLE>Hello World Example</TITLE></HEAD>
<BODY>Hello, World!</BODY></HTML>
>>>
```

Filling a Template: Part I

```
t = Template(templateDef,
    searchList=[namespage])
```

- Generated a subclass of Template
  - ✳ “compiling the template definition”
  - ✳ “generated template class”
- Instantiated the subclass
  - ✳ “template instance”
- __str__() renders the filled template

Instances Respond to Namespace Mutations

```
>>> nameSpace['title'] = 'Mutated Example'
>>> nameSpace['contents'] = 'Hiya, Planet Earth!'
>>> print t

<HTML><HEAD><TITLE>Mutated Example</TITLE></HEAD>
<BODY>Hiya, Planet Earth!</BODY></HTML>
>>>
```

Filling a Template: Part 2

```
>>> t2 = Template.Template(templateDef)
>>> t2.title = 'Templates R Fun'
>>> t2.contents = 'Howdy, TriZPUG!'
>>> print t2
<HTML><HEAD><TITLE>Templates R Fun</TITLE></HEAD>
<BODY>Howdy, TriZPUG!</BODY></HTML>
>>>
```

The generated subclass instance may be decorated with instance attributes named for placeholders in the template definition

Filling a Template: Part 3

```
>>> nameSpace = {'title': 'Templates are Instances',
...              'contents': 'Of Generated Subclasses'}
>>> dynamicTmpl = Template.Template.compile(templateDef)
>>> t3 = dynamicTmpl(searchList=[nameSpace])
>>> print t3

<HTML><HEAD><TITLE>Templates are Instances</TITLE></HEAD>
<BODY>Of Generated Subclasses</BODY></HTML>
>>>
```

Filling a Template: Part 3

```
tmpl = Template.compile(tDef)
```

- `compile` is a classmethod of `Template`
- returns the generated subclass
- does not instantiate the generated subclass

Compile and Instantiate

```
t = Template('TriZPUG is $awesome')
```

- Pass the template definition as a string

Compile and Instantiate

```
t = Template(file='trizpug.tmpl')
```

- Read the template definition from a file
- The file path is passed as a string

Compile and Instantiate

```
t = Template(file= \
                  open('trizpug.tmpl'))
```

- Read the template definition from a file
- A file-like object is passed

Compile, Instantiate, Fill

```
t = Template('TriZPUG is $awesome',
                    searchList=[
                        {'awesome': 'hotness'}])
```

- Pass the template definition as a string

Compile, Instantiate, Fill

```
t = Template(file='trizpug.tmpl',
             searchList=[
                 {'awesome': 'hotness'}])
```

- Read the template definition from a file
- The file path is passed as a string

Compile, Instantiate, Fill

```
t = Template(file= \
                  open('trizpug.tmpl'),
                  searchList=[
                  {'awesome': 'hotness'}])
```

- Read the template definition from a file
- A file-like object is passed

Compiling Template

Contents of `trizpug.tmpl`:

```
TriZPUG is $awesome
```

Compiling Template

```
> bin/cheetah compile trizpug.tmpl
Compiling trizpug.tmpl -> trizpug.py
> ls
bin include lib trizpug.py trizpug.tmpl
>
```

Compiling Template

Contents of `trizpug.py`:

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

...

from Cheetah.Template import Template

...

class trizpug(Template):

...
```

Import the Template Module

```
>>> from trizpug import trizpug
>>> t = trizpug()
>>> t.awesome = 'hotness'
>>> print t
TriZPUG is hotness
>>>
```

Import the Template Module

```
>>> from trizpug import trizpug
>>> t = trizpug(searchList= \
        [{'awesome':'hotness'}])
>>> print t
TriZPUG is hotness
>>>
```

The `respond()` Method

```
>>> from trizpug import trizpug
>>> t = trizpug()
>>> t.awesome = 'hotness'
>>> t.respond()
u'TriZPUG is hotness'
>>>
```

Running the Template Module

```
> bin/python trizpug.py --help
Cheetah 2.2.1 template module command-
line interface

Usage

-----

  trizpug.py [OPTION]

...
```

Filling a Template

Contents of `trizpug.tmpl`:

```
#def awesome
hotness
#end def
TriZPUG is $awesome
```

Filling a Template

```
> bin/cheetah fill trizpug.tmpl
Filling trizpug.tmpl -> trizpug.html
> ls
bin include lib trizpug.html trizpug.tmpl
>
```

Filling a Template

Contents of `trizpug.html`:

```
TriZPUG is hotness
```

Compiling and Filling

```
> bin/cheetah compile trizpug.tmpl
Compiling trizpug.tmpl -> trizpug.py
> bin/python
>>> from trizpug import trizpug
>>> t = trizpug()
>>> t.respond()
u'TriZPUG is hotness'
>>>
```

Inheriting from a Template

Contents of `trizpug.tmpl`:

```
TriZPUG is $awesome
```

Inheriting from a Template

Contents of **zpugdc.tmpl**:

```
#from trizpug import trizpug
#extends trizpug
#def awesome
hotness
#end def
```

Inheriting from a Template

```
> bin/cheetah compile trizpug.tmpl
Compiling trizpug.tmpl -> trizpug.py
> bin/cheetah compile zpugdc.tmpl
Compiling zpugdc.tmpl -> zpugdc.py
> bin/python
>>> from zpugdc import zpugdc
>>> z = zpugdc()
>>> z.respond()
u'TriZPUG is hotness'
>>>
```

Inheriting from a Template

```
> bin/python zpugdc.py
TriZPUG is hotness
>
```

Inheriting from a Template

```
>>> from trizpug import trizpug
>>> class zpugdc(trizpug):
...     def awesome(self):
...         return 'hotness'
...
>>> z = zpugdc()
>>> z.respond()
u'TriZPUG is hotness'
>>>
```

Template Comments

```
>>> templateDef = """
... <HTML><HEAD><TITLE>$title</TITLE></HEAD>
... ## This is a single line Cheetah comment.
... ## It won't appear in the output.
... #* This is a multiline Cheetah comment.
...    It won't appear in the output, either.
... *#
... <BODY>$contents</BODY></HTML>"""
>>>
```

Template Comments

```
>>> templateDef = """
... <HTML><HEAD><TITLE>$title</TITLE></HEAD>
... \## This is NOT a single line Cheetah comment.
... \## It WILL appear in the output.
... \#* This is NOT a multiline Cheetah comment.
...    It WILL appear in the output, either.
... *#
... <BODY>$contents</BODY></HTML>"""
>>>
```

Template Comments

```
>>> templateDef = """
... #*doc: This text will be added to
...         your respond method's docstring *#
... """
>>>
```

Template Comments

```
>>> templateDef = """
... ##doc: This, too!
>>>
```

Template Comments

```
>>> templateDef = """
... #*doc-method: This text will be added to
...         your respond method's docstring *#
... """
>>>
```

Template Comments

```
>>> templateDef = """
... #*doc-class: This text will be added to
...         the generated class docstring *#
... """
>>>
```

Template Comments

```
>>> templateDef = """
... #*doc-module: This text will be added to
...         the compiled module docstring *#
... """
>>>
```

#echo Directive

```
>>> templateDef = """
... Check #echo dir() # out!
... """
>>>
```

- `#echo` evaluates an expression into the output
- Directives can be in the middle of a line
- Another # ends a directive

`#silent` Directive

```
>>> templateDef = """
... Check #silent dir() # out!
... """
>>>
```

- `#silent` evaluates an expression
- The value of the expression is discarded

One-line `#if` Directive

```
>>> templateDef = """
... #if __name__ == '__main__' then 'script' else 'module'#
... """
>>>
```

- `#if` selects one of two expressions to include in the output
- There must be both **`then`** and **`else`**

`#raw` Directive

```
>>> templateDef = """
... #raw
... $these $placeholders $will $not $be $parsed
... #echo 'neither will this directive'
... #end raw
... """
>>>
```

#include Directive

```
>>> templateDef = """
... #* The following directive includes
...    the contents of a file name trizpug.txt
...    which will then be parsed for placeholders
...    and directives *#
... #include 'trizpug.txt'
... """
>>>
```

#include Directive

```
>>> templateDef = """
... #* The following directive includes
...    the contents of a file name trizpug.txt
...    which will then NOT be parsed for placeholders
...    and directives *#
... #include raw 'trizpug.txt'
... """
>>>
```

#import Directive

```
>>> templateDef = """
... #import math
... $math.pi
... """
>>> t = Template(templateDef)
>>> print t

3.14159265359

>>>
```

Note the use of dot notation on a filled placeholder

#from Directive

```
>>> templateDef = """
... #from math import e
... $e
... """
>>> t = Template(templateDef)
>>> print t

2.71828182846

>>>
```

#extends Directive

```
>>> templateDef = """
... #extends trizpug
... ## implicitly does \#from trizpug import trizpug
... ## and causes your Template to inherit from trizpug
...
... #extends Cheetah.Templates.SkeletonPage
... #* implicitly does
... \#from Cheetah.Templates.SkeletonPage import SkeletonPage
... and causes your Template to inherit from SkeletonPage
... *#
... """
>>>
```

Your templates may inherit from other Templates or subclasses of other Templates

`#set` Directive

```
>>> templateDef = """
... #set $awesome = 'hotness'
... TriZPUG is $awesome"""
...
>>> t = Template(templateDef)
>>> print t

TriZPUG is hotness

>>>
```

`#set` fills a *local variable* that is searched when filling placeholders

#for Directive

```
>>> templateDef = """
... #for $country in ['England','France','Germany']
... I see $country
... #end for"""
>>> t = Template(templateDef)
>>> print t

I see England
I see France
I see Germany

>>>
```

#for iterates over a sequence

#repeat Directive

```
>>> templateDef = """
... #repeat 3
... TriZPUG!
... #end repeat
... """
>>> t = Template(templateDef)
>>> print t

TriZPUG!
TriZPUG!
TriZPUG!

>>>
```

#repeat loops a definite number of times

#while Directive

```
>>> templateDef = """
... #set $x = 0
... #while $x < 3
... TriZPUG!
... #set $x = $x + 1
... #end while"""
>>> t = Template(templateDef)
>>> t.respond()
u'\nTriZPUG!\nTriZPUG!\nTriZPUG!\n'
>>>
```

#while loops until a condition is false

#if Directives

```
>>> templateDef = """
... #set $pugs = {'USA':'TriZPUG','Germany':'DZUG','Canada':'PyGTA'}
... #for $pug in $pugs.items
... #if $pug[0] == 'USA'
... $pug[1]
... #elif $pug[0] == 'Germany'
... MacYet!
... #else
... Something completely different
... #end if
... #end for
... """
>>> t = Template(templateDef)
>>> t.respond()
u'\nSomething completely different\nMacYet!\nTriZPUG\n'
>>>
```

Placeholder Search Order

- Local variables (`#set`, `#for`, etc.)
- Global variables (`#set global`)
- `searchList` passed to `Template`
- Instance attributes (`#def`, `#attr`, etc.)
- Python globals (`#import`, `#from`, etc.)
- Python builtins (`None`, `max`, etc.)

CHEETAH

The Python-Powered
Template Engine

Much more (filters, caching, error handling, etc.) at:
http://cheetahtemplate.org