

Google App Engine Using Templates

Charles Severance and Jim Eng
csev@umich.edu jimeng@umich.edu

Textbook: Using Google App Engine, Charles Severance

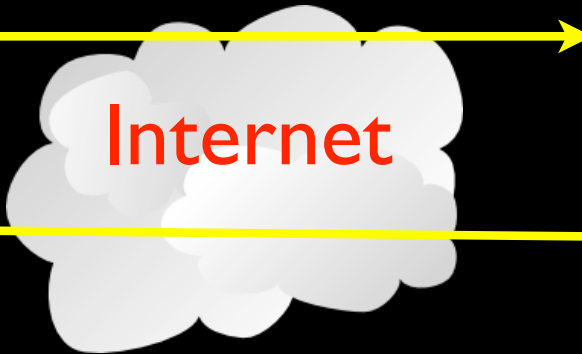
The logo for open.michigan, featuring the text "open.michigan" in a lowercase, sans-serif font. The "o" in "open" is stylized with a dot. Below the text are several thin, wavy blue lines that create a sense of motion or a landscape.

Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.

<http://creativecommons.org/licenses/by/3.0/>.

Copyright 2009, Charles Severance and Jim Eng





HTML JavaScript
AJAX CSS

HTTP Request
Response GET
POST

Python Data Store
Templates memcache
WebApp MVC

Templates

- While we could write all of the HTML into the response using `self.response.out.write()`, we really prefer not to do this
- Templates allow us to separately edit HTML files and leave little areas in those files where data from Python gets dropped in
- Then when we want to display a view, we process the template to produce the HTTP Response

<http://docs.djangoproject.com/en/dev/ref/templates/builtins/?from=olddocs>

Google App Engine Basic Templates

ae-04-template

www.appenginelearn.com

```
class MainHandler(webapp.RequestHandler):

    formstring = "<form method='post' action='/'>
<p>Enter Guess:
<input type='text' name='guess'></p>
<p><input type='submit'></p>
</form>"

    def get(self):
        self.response.out.write('<p>Good luck!</p>\n')
        self.response.out.write(self.formstring)

    def post(self):
        stguess = self.request.get('guess')
        logging.info('User guess='+stguess)
        try:
```

YUCK!!

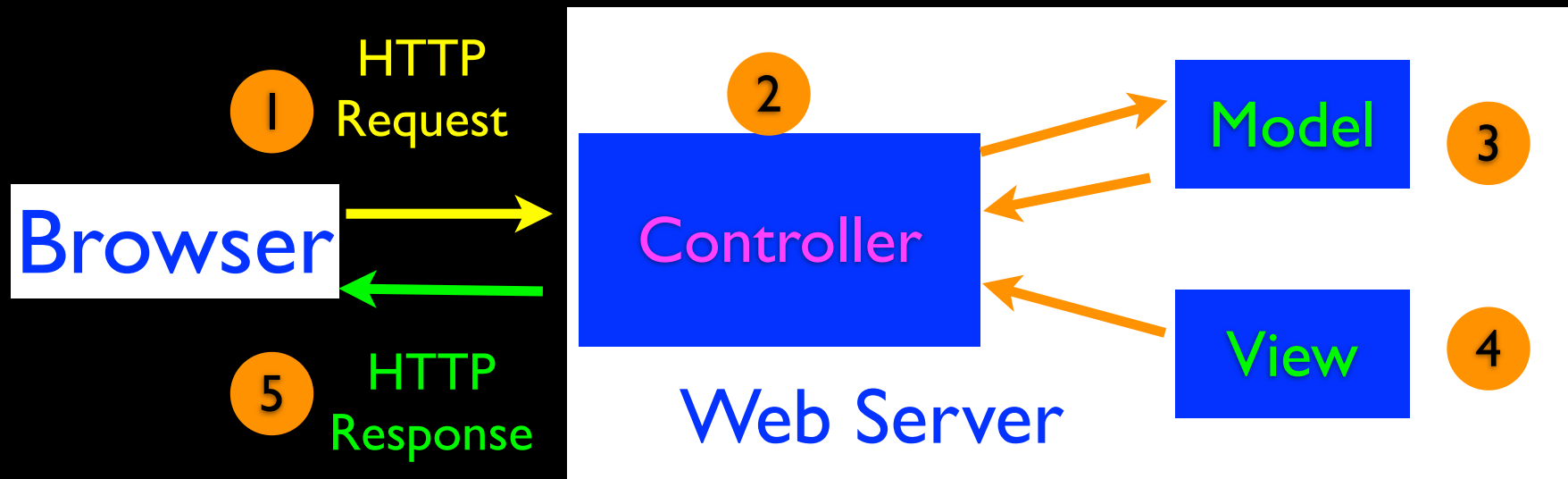
Python strings are a **lousy** way to store and edit HTML. Your code gets obtuse and nasty. Lets move the HTML into a separate file.

Separation of Concerns

- A well written App Engine Application has no HTML in the Python code - it processes the input data, talks to databases, makes lots of decisions, figures out what to do next and then
- Grabs some HTML from a template - replacing a few selected values in the HTML from computed data - and viola! We have a response.

Terminology

- We name the three basic functions of an application as follows
 - **Controller** - The Python code that does the thinking and decision making
 - **View** - The HTML, CSS, etc. which makes up the look and feel of the application
 - **Model** - The persistent data that we keep in the data store



MVC

- We call this pattern the “**Model - View - Controller**” pattern (or **MVC** for short)
- It is a very common pattern in web applications - not just Google Application Engine
 - Ruby on Rails
 - Spring MVC
- We will meet the “Model” later - for now we will work with the View and Controller

Back to: Templates

- A template is **mostly HTML** but we have some little syntax embedded in the HTML to drop in bits of data at run-time
- The controller computes the “bits” and gives them to the “Render Engine” to put into the template.

A Simple Template

```
<p>{{ hint }}</p>  
<form method="post" action="/">  
<p>Enter Guess: <input type="text" name="guess"/></p>  
<p><input type="submit"></p>  
</form>
```

Mostly **HTML** - with a little
place to drop in **data from**
the Controller.

In The Controller

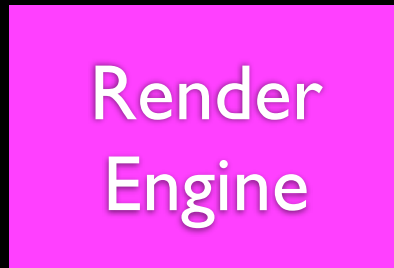
- In the controller, we prepare a **Python Dictionary object** with the data for the template and call the “Render Engine”

```
outstr = template.render(filepath, { 'hint' : 'Too low' })
```

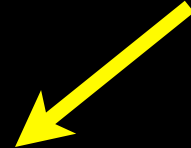
The Render Engine takes two parameters (1) the **path to a template file**, and (2) a **Python dictionary with key value pairs** of the data areas in the template.

Render
Data

Template



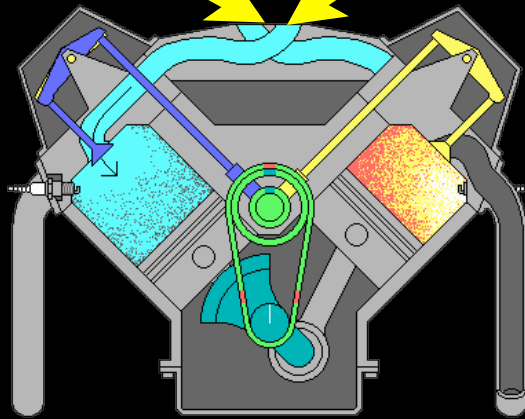
Rendered
Output



{ 'hint' : 'Too Low' }

<p>{{ hint }}</p>
<form method="post"
action="/">
...

V-8 Render Engine



<p>Too Low</p>
<form method="post"
action="/">
...

Template Pattern

- We store templates in a folder called “`templates`” under the main application directory to keep the templates (views) separate from the Python code (controller)
- We need to load the template from the right place in our Python code (it is a little ugly...)

```
filepath = os.path.join(os.path.dirname(__file__), 'templates/index.htm')  
outstr = template.render(filepath, { 'hint' : 'Too low'})
```



```
def post(self):
    stguess = self.request.get('guess')
    guess = int(stguess)
    if guess == 42:
        msg = 'Congratulations'
    elif guess < 42:
        msg = 'Your guess is too low'
    else:
        msg = 'Your guess is too high'
```

We read the guess, convert it to an integer, check if it is right or wrong, setting a message variable and then passing some data into a template to be rendered.

```
temp = os.path.join(os.path.dirname(__file__), 'templates/guess.htm')
outstr = template.render(temp, {'hint': msg, 'oldguess': stguess})
self.response.out.write(outstr)
```

Good luck!

Enter Guess:

Controller and View

Your Guess

Your guess is too low

Enter Guess:

```
def post(self):
    stguess = self.request.get('guess')
    guess = int(stguess)
    if guess == 42:
        msg = 'Congratulations'
    elif guess < 42:
        msg = 'Your guess is too low'
    else:
        msg = 'Your guess is too high'

    temp = os.path.join(os.path.dirname(__file__),
        'templates/guess.htm')
    outstr = template.render(temp, {'hint': msg,
        'oldguess': stguess})
    self.response.out.write(outstr)
```

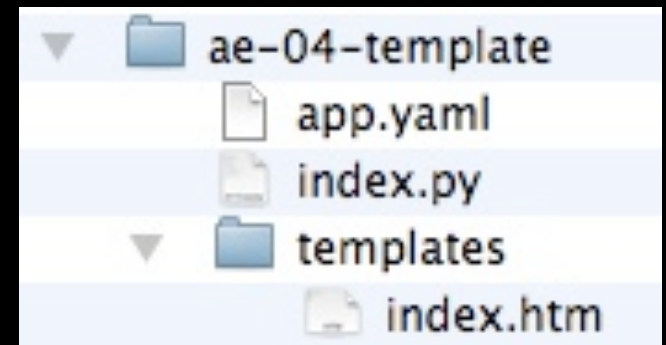
```
<p>Your Guess: {{ oldguess }}</p>
<p>{{ hint }}</p>
<form method="post" action="/">
<p>Enter Guess: <input type="text"
name="guess"/></p>
<p><input type="submit"></p>
</form>
```

Controller

View

Application Structure

- We keep the `app.yaml` and `index.py` files in the main application folder and the templates are stored in a folder called “`templates`”
- This is not a **rule** - just a pattern that it makes it easier to look at someone else’s code



Template Summary

- We separate the logic of our program (Controller) from the HTML bits of the program (View) to keep things cleaner and more organization
- We use the Google templating engine to read the templates and substitute **bits of computed data** into the resulting HTML

`<p>{{ hint }}</p>`

`<form method="post"
action="/">`

+ { 'hint' : 'Too Low' } =

`<p>Too Low</p>`

`<form method="post"
action="/">`

...

...

Several Templates

Program: ae-05-templates

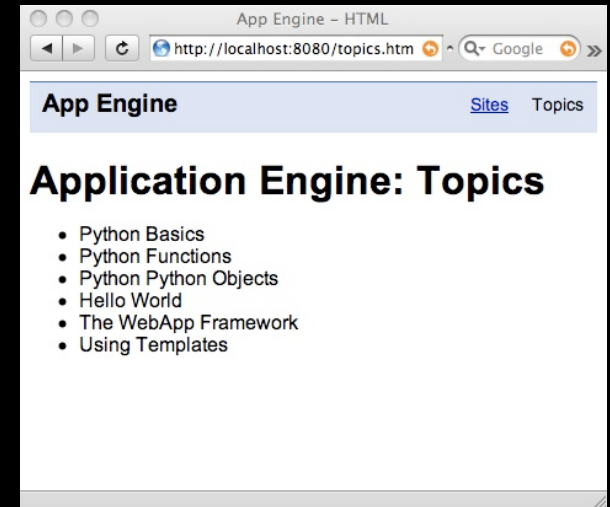
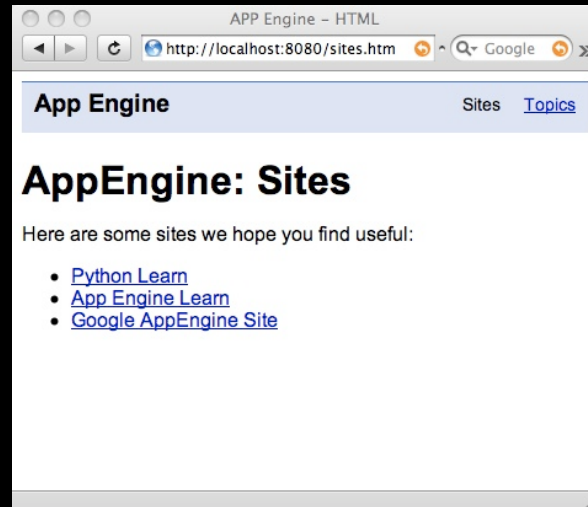
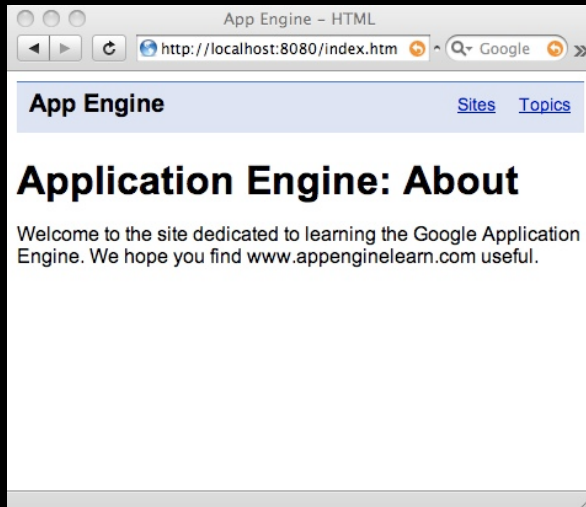
www.appenginelearn.com

Real Applications

- Real applications have lots of handlers and **lots of templates**
- In this section we start to look at techniques for managing and organizing templates

<http://docs.djangoproject.com/en/dev/ref/templates/builtins/?from=olddocs>

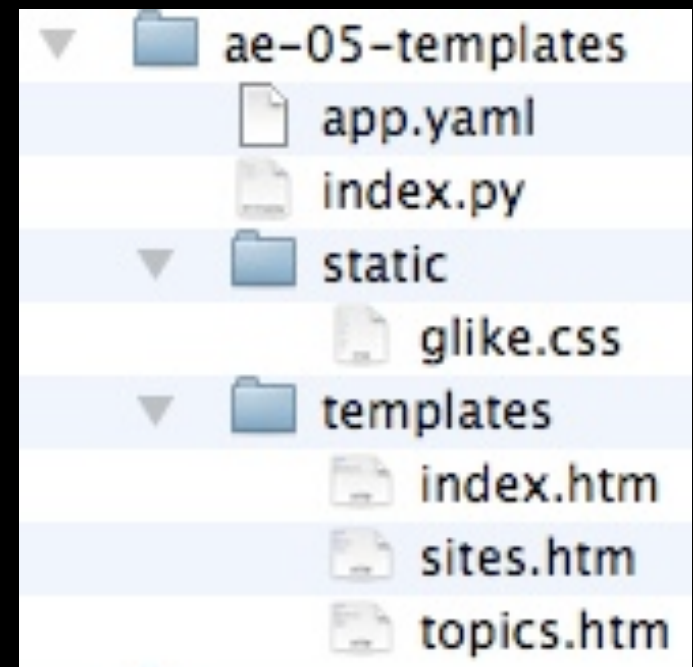
Our Application



Our Application has three pages - no forms, and a bit of CSS to make the navigation pretty and light blue. It is mostly a static site.

Application Layout

- There are three templates in the templates directory
- The CSS file is in the static directory - this is a special directory



Looking at app.yaml

- The app.yaml file has a new handler for **static data** **which does not change** like images, CSS, javascript libraries, etc
- Google serves these “**read-only**” files **very** efficiently
- Identifying them as static can save you money

```
application: ae-05-templates
version: 1
runtime: python
api_version: 1

handlers:
- url: /static
  static_dir: static

- url: /*
  script: index.py
```

Looking at app.yaml

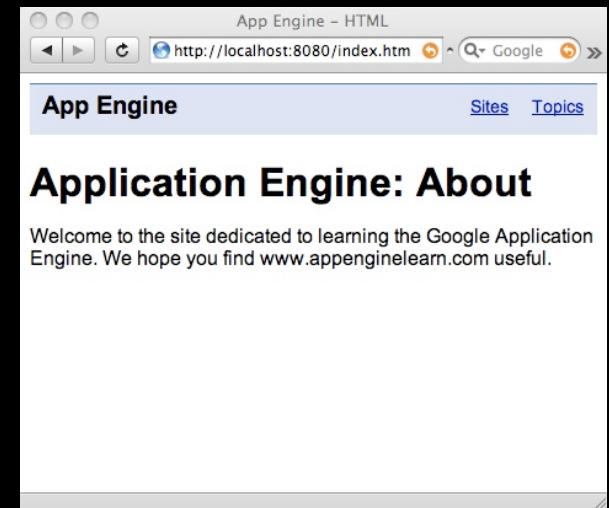
- The handlers in the app.yaml file are checked in order
- First it looks at the url to see if it starts with “/static”
- The last URL is a catch-all - send everything to the controller (`index.py`)

```
application: ae-05-templates
version: 1
runtime: python
api_version: 1

handlers:
- url: /static
  static_dir: static

- url: /*
  script: index.py
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>App Engine - HTML</title>
  <link href="/static/glike.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <div id="header">
    <h1><a href="index.htm" class="selected">App Engine</a></h1>
    <ul class="toolbar">
      <li><a href="sites.htm">Sites</a></li>
      <li><a href="topics.htm" >Topics</a></li>
    </ul>
  </div>
  <div id="bodycontent">
    <h1>Application Engine: About</h1>
    <p>
      Welcome to the site dedicated to
      learning the Google Application Engine.
      We hope you find www.appenginelearn.com useful.
    </p>
  </div>
</body>
</html>
```



The templates are just flat HTML. The only real App Engine change is that the CSS file is coming from “/static”

Controller Code

- The controller code is going to be very general
- It will look at the path on the URL and try to find a template of that name - if that fails, render the `index.htm` template

`http://localhost:8080/topics.htm`

Path


For this URL, the path is `/topics.htm`

```
class MainHandler(webapp.RequestHandler):
```

```
    def get(self):  
        path = self.request.path
```

`http://localhost:8080/topics.htm`



```
    try:
```

```
        temp = os.path.join(os.path.dirname(__file__), 'templates' + path)  
        outstr = template.render(temp, { })  
        self.response.out.write(outstr)
```

```
    except:
```

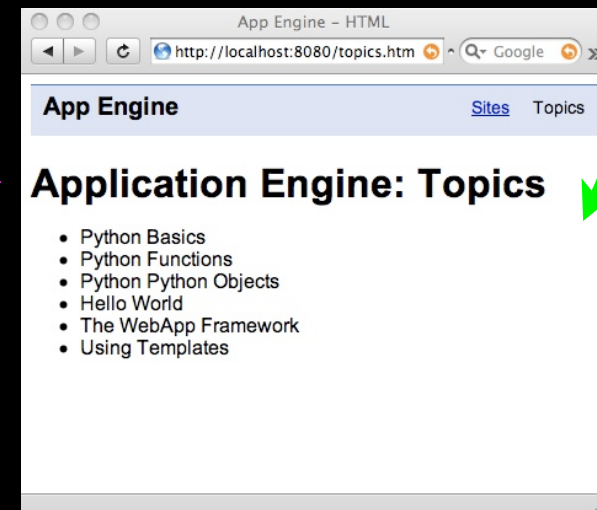
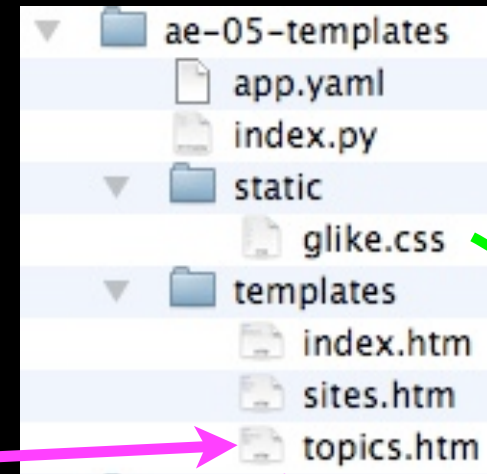
```
        temp = os.path.join(os.path.dirname(__file__), 'templates/index.htm')  
        outstr = template.render(temp, { })  
        self.response.out.write(outstr)
```

If all else fails, render `templates/index.htm`

Note that we are **not** passing any data to the templates.

<http://localhost:8080/topics.htm>

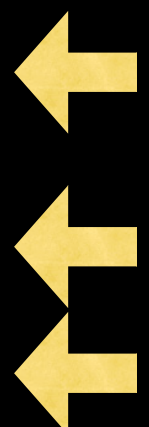
```
path = self.request.path
temp = os.path.join(... 'templates' + path)
outstr = template.render(temp, { })
self.response.out.write(outstr)
```



The browser also does a GET request for </static/glike.css>

In the Log....

```
Terminal — Python — 90x21
Python bash bash
charles-severances-macbook-air:apps csev$ dev_appserver.py ae-05-templates/
INFO      2008-10-21 23:54:42,058 appcfg.py] Server: appengine.google.com
INFO      2008-10-21 23:54:42,079 appcfg.py] Checking for updates to the SDK.
INFO      2008-10-21 23:54:42,248 appcfg.py] The SDK is up to date.
WARNING   2008-10-21 23:54:42,249 datastore_file_stub.py] Could not read datastore data from /var/folders/jW/jW3AfyxcGF09fub-nVQ5uE+++TM/-Tmp-/dev_appserver.datastore
WARNING   2008-10-21 23:54:42,250 datastore_file_stub.py] Could not read datastore data from /var/folders/jW/jW3AfyxcGF09fub-nVQ5uE+++TM/-Tmp-/dev_appserver.datastore.history
INFO      2008-10-21 23:54:42,321 dev_appserver_main.py] Running application ae-05-templates on port 8080: http://localhost:8080
INFO      2008-10-21 23:54:45,803 dev_appserver.py] "GET /index.htm HTTP/1.1" 200 -
INFO      2008-10-21 23:54:45,922 dev_appserver_index.py] Updating /Users/csev/Desktop/teach/a539-f08/apps/ae-05-templates/index.yaml
INFO      2008-10-21 23:54:45,949 dev_appserver.py] "GET /static/glike.css HTTP/1.1" 200 -
INFO      2008-10-21 23:54:47,400 dev_appserver.py] "GET /sites.htm HTTP/1.1" 200 -
INFO      2008-10-21 23:54:47,422 dev_appserver.py] "GET /static/glike.css HTTP/1.1" 200 -
INFO      2008-10-21 23:54:49,445 dev_appserver.py] "GET /topics.htm HTTP/1.1" 200 -
INFO      2008-10-21 23:54:49,469 dev_appserver.py] "GET /static/glike.css HTTP/1.1" 200 -
█
```



Extending Base Templates

Program: ae-06-templates

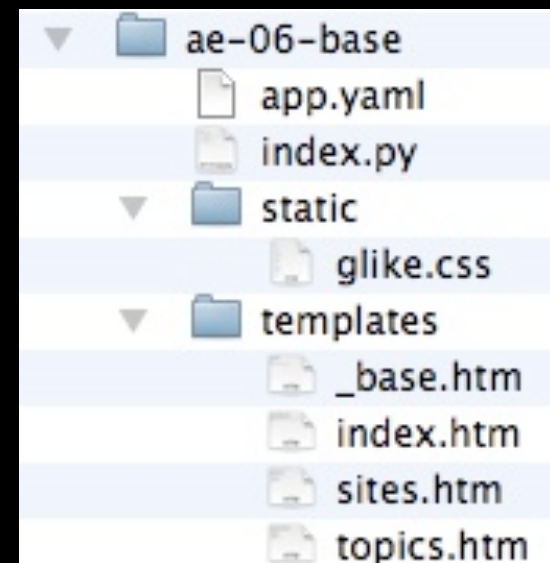
www.appenginelearn.com

Base Templates

- When building web sites there is a great deal of common material across pages
 - head
 - navigation
- Often only a small amount of information changes between pages

Application Layout

- This is the same as the previous application except we refactor the templates, putting the common material into the file `_base.htm`
- We reuse the `_base.htm` content in each of the other templates



```

<head>
  <title>App Engine - HTML</title>
  <link href="/static/glike.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <div id="header">
    <h1><a href="index.htm" class="selected">
      App Engine</a></h1>
    <ul class="toolbar">
      <li><a href="sites.htm">Sites</a></li>
      <li><a href="topics.htm" >Topics</a></li>
    </ul>
  </div>
  <div id="bodycontent">
    <h1>Application Engine:About</h1>
    <p>
      Welcome to the site dedicated to
      learning the Google Application Engine.
      We hope you find www.appenginelearn.com useful.
    </p>
  </div>
</body>
</html>

```

These files are nearly identical. And we have lots of files like this.

```

<head>
  <title>App Engine - HTML</title>
  <link href="/static/glike.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <div id="header">
    <h1><a href="index.htm" >
      App Engine</a></h1>
    <ul class="toolbar">
      <li><a href="sites.htm">Sites</a></li>
      <li><a href="topics.htm" class="selected">Topics</a></li>
    </ul>
  </div>
  <div id="bodycontent">
    <h1>Application Engine:Topics</h1>
    <ul>
      <li>Python Basics</li>
      <li>Python Functions</li>
      <li>Python Python Objects</li>
      <li>Hello World</li>
      <li>The WebApp Framework</li>
      <li>Using Templates</li>
    </ul>
  </div>
</body>
</html>

```

A Base Template

- We create a base template that contains the **material that is common** across the pages and leave a little place in the base template to put in the bits that change

```
<head>
  <title>App Engine - HTML</title>
  <link href="/static/glike.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <div id="header">
    <h1><a href="index.htm" class="selected">
      App Engine</a></h1>
    <ul class="toolbar">
      <li><a href="sites.htm">Sites</a></li>
      <li><a href="topics.htm" >Topics</a></li>
    </ul>
  </div>
  <div id="bodycontent">
    <!-- Application Engine -->
    <!-- Replace this
    Welcome to this site dedicated to
    learning the Google Application Engine.
    We hope you find www.appenginelearn.com useful.
    -->
  </div>
</body>
</html>
```

_base.htm

index.htm

```

<head>
  <title>App Engine - HTML</title>
  <link href="/static/glike.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <div id="header">
    <h1><a href="index.htm" class="selected">
      App Engine</a></h1>
    <ul class="toolbar">
      <li><a href="sites.htm">Sites</a></li>
      <li><a href="topics.htm" >Topics</a></li>
    </ul>
  </div>
  <div id="bodycontent">
    {% block bodycontent %}
      Replace this
    {% endblock %}
  </div>
</body>
</html>

```

`_base.htm`

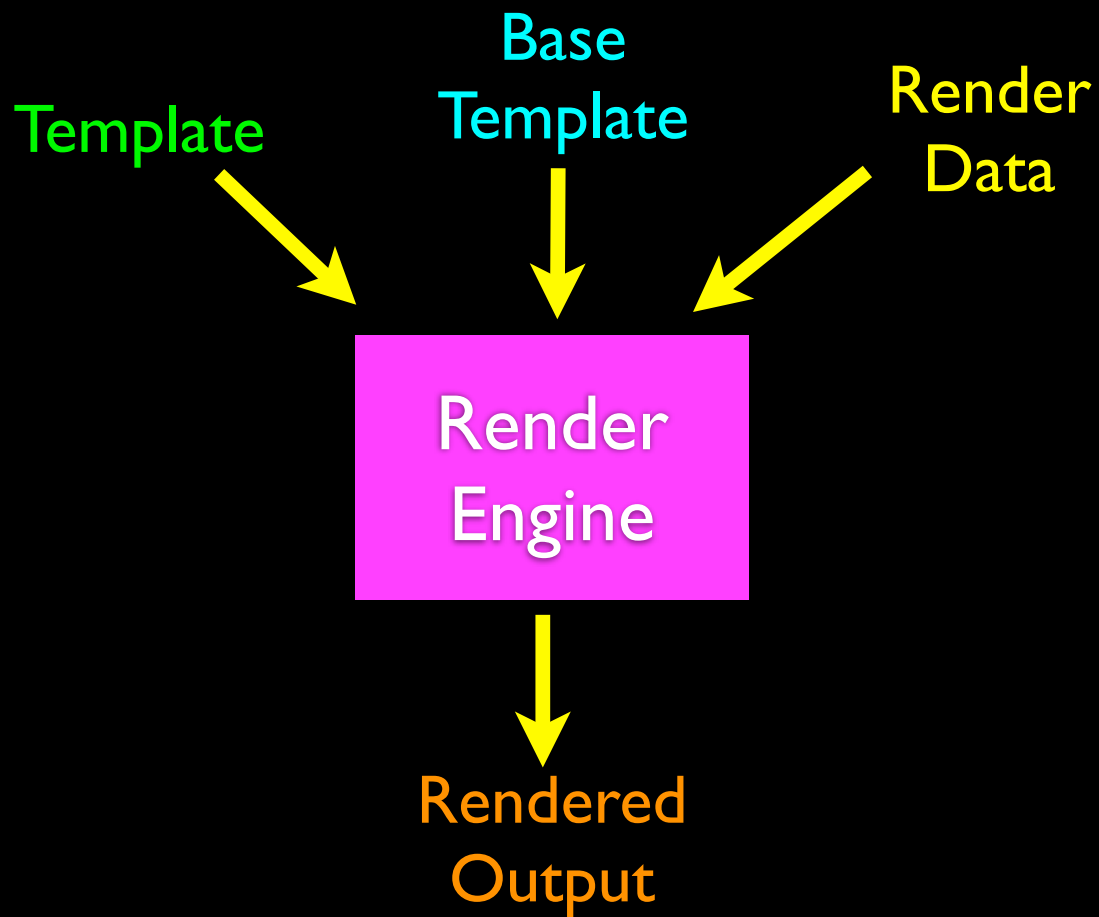
The “**extends**” indicates that this page is to “start with” `_base.htm` as its overall text and replace the **bodycontent** block in `_base.htm` with the **given text**.

```

{% extends "_base.htm" %}
{% block bodycontent %}
  <h1>Application Engine:About</h1>
  <p>
    Welcome to the site dedicated to
    learning the Google Application Engine.
    We hope you find www.appenginelearn.com useful.
  </p>
{% endblock %}

```

`index.htm`



```

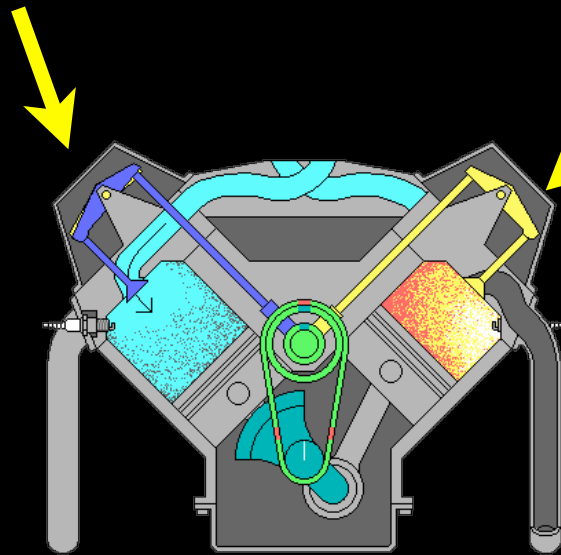
{% extends "_base.htm" %}
{% block bodycontent %}
  <h1>Application Engine:About</h1>
  <p>
    Welcome to the site dedicated to
    learning the Google Application Engine.
    We hope you find www.appenginelearn.com useful.
  </p>
{% endblock %}

```

```

<head>
  <title>App Engine - HTML</title>
</head>
<body>
  </div>
  <div id="bodycontent">
    {% block bodycontent %}
      Replace this
    {% endblock %}
  </div>
</body>
</html>

```



V-8 Render Engine

{ 'dat' : 'Fun Stuff' }

```

...
<div id="bodycontent">
  <h1> ....
</div>
...

```


Extending a Base Template

- This capability to **extend** a base template is just part of the standard template render processing
- The template which is rendered is “**index.htm**”
- The render engine reads through **index.htm**. It sees the **extend** directive and goes to get the content of **_base.htm** as the starting point for **index.htm**

```
{% extends "_base.htm" %}  
{% block bodycontent %}  
    <h1>Application Engine:About</h1 >  
    ...  
{% endblock %}
```

Making Navigation Look Nice

Program: ae-06-templates

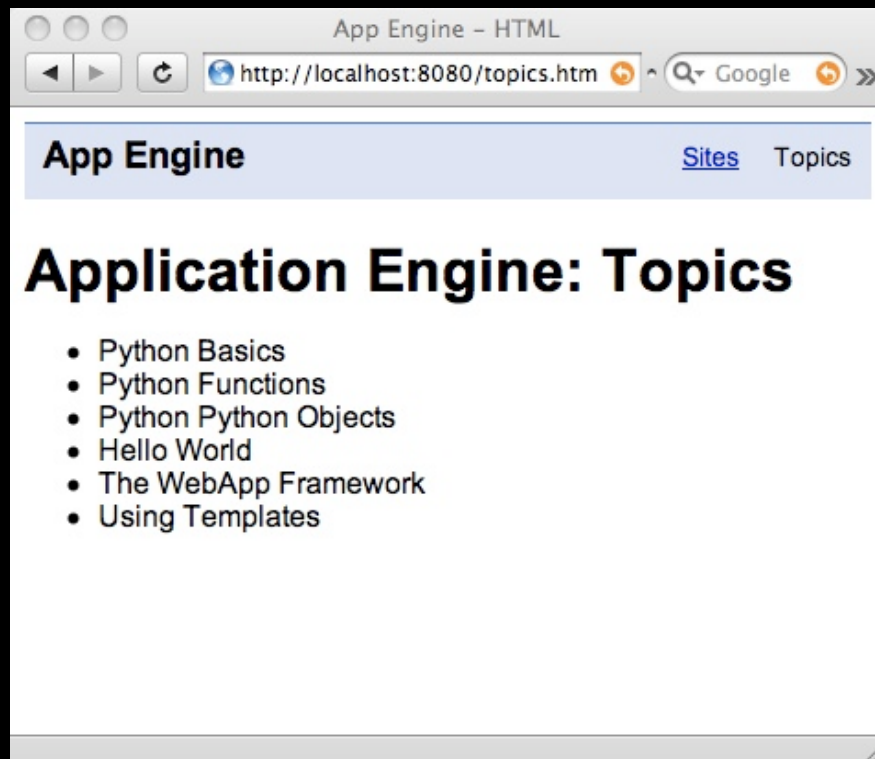
www.appenginelearn.com

Navigation Issues

- As we navigate between pages, we want the look of the "current" page to change color or provide some indication which page we are on.
- This is usually done with a CSS class on the tag

```
<ul class="toolbar">  
  <li><a href="sites.htm">Sites</a></li>  
  <li><a href="topics.htm" class="selected">Topics</a></li>  
</ul>
```

```
<ul class="toolbar">
  <li><a href="sites.htm">Sites</a></li>
  <li><a href="topics.htm" class="selected">Topics</a></li>
</ul>
```



In `topics.htm`, the **style sheet** changes the Topics link to be Black and not underlined.

```
a.selected {
  color: black;
  text-decoration: none;
}
```

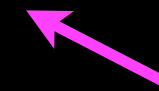
Problem

- In this situation - the link that is **selected changes** between pages
- We need to put `class="selected"` on `<a>` tag for the current page but not for the other pages

Solution

- We pass the current path for the page into the template as a render parameter
- In the template we **check** the current path and only emit the class="selected" when the path is the current page

`http://localhost:8080/topics.htm`



Path

```
class MainHandler(webapp.RequestHandler):
```

```
    def get(self):
```

```
        path = self.request.path
```

```
        try:
```

```
            temp = os.path.join(os.path.dirname(__file__), 'templates' + path)
```

```
            outstr = template.render(temp, { 'path': path })
```

```
            self.response.out.write(outstr)
```

```
        except:
```

```
            temp = os.path.join(os.path.dirname(__file__), 'templates/index.htm')
```

```
            outstr = template.render(temp, { 'path': path })
```

```
            self.response.out.write(outstr)
```

_base.htm

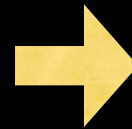
```
<ul class="toolbar">
  <li><a href="sites.htm"
    {% ifequal path '/sites.htm' %}
      class="selected"
    {% endifequal %}
    >Sites</a></li>
  <li><a href="topics.htm"
    {% ifequal path '/topics.htm' %}
      class="selected"
    {% endifequal %}
    >Topics</a></li>
</ul>
```

For each of the links, if the path matches, we emit class="selected" otherwise we do not.

Conditional HTML generation.

_base.htm

```
<ul class="toolbar">
  <li><a href="sites.htm"
    {% ifequal path '/sites.htm' %}
      class="selected"
    {% endifequal %}
    >Sites</a></li>
  <li><a href="topics.htm"
    {% ifequal path '/topics.htm' %}
      class="selected"
    {% endifequal %}
    >Topics</a></li>
</ul>
```

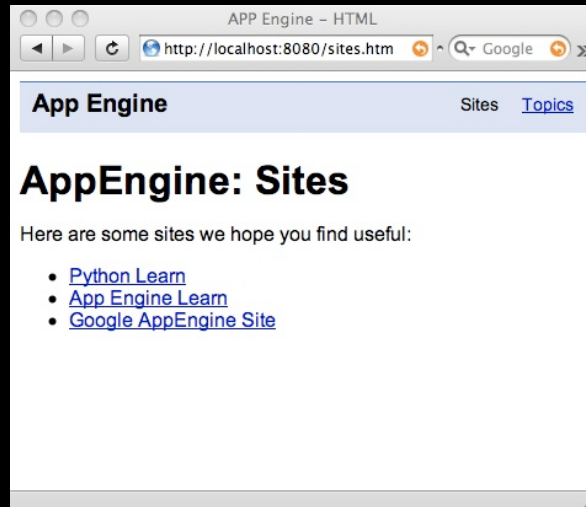
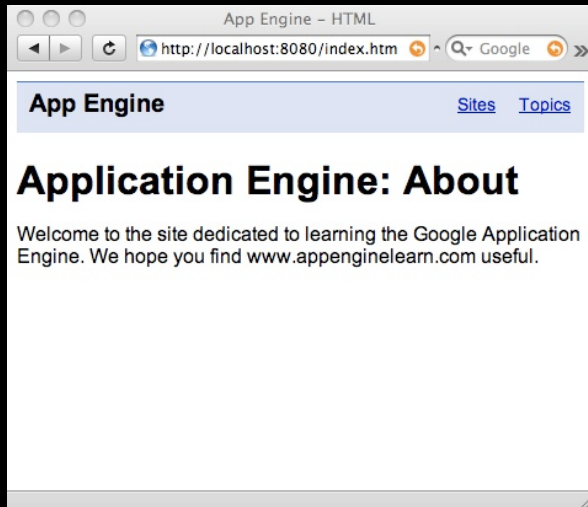


topics.htm (rendered)

```
<ul class="toolbar">
  <li><a href="sites.htm"
    >Sites</a></li>
  <li><a href="topics.htm"
    class="selected"
    >Topics</a></li>
</ul>
```

The **path** variable comes from the Python code.

Our Application



Program: ae-06-templates

More on Templates



- This is only scratching the surface of templates
- The Google Application Engine templating language is taken from the **django** application
- You can read further in the **django** documentation

<http://docs.djangoproject.com/en/dev/ref/templates/builtins/?from=olddocs>

Summary

- We can use the ability to create a base template and then extend it in our regular templates to reduce the amount of repeated HTML code in templates.
- We can even make pretty navigation links which change based on which page is the current page
- When we don't have to repeat the same code over and over - it is easy to make changes without breaking things