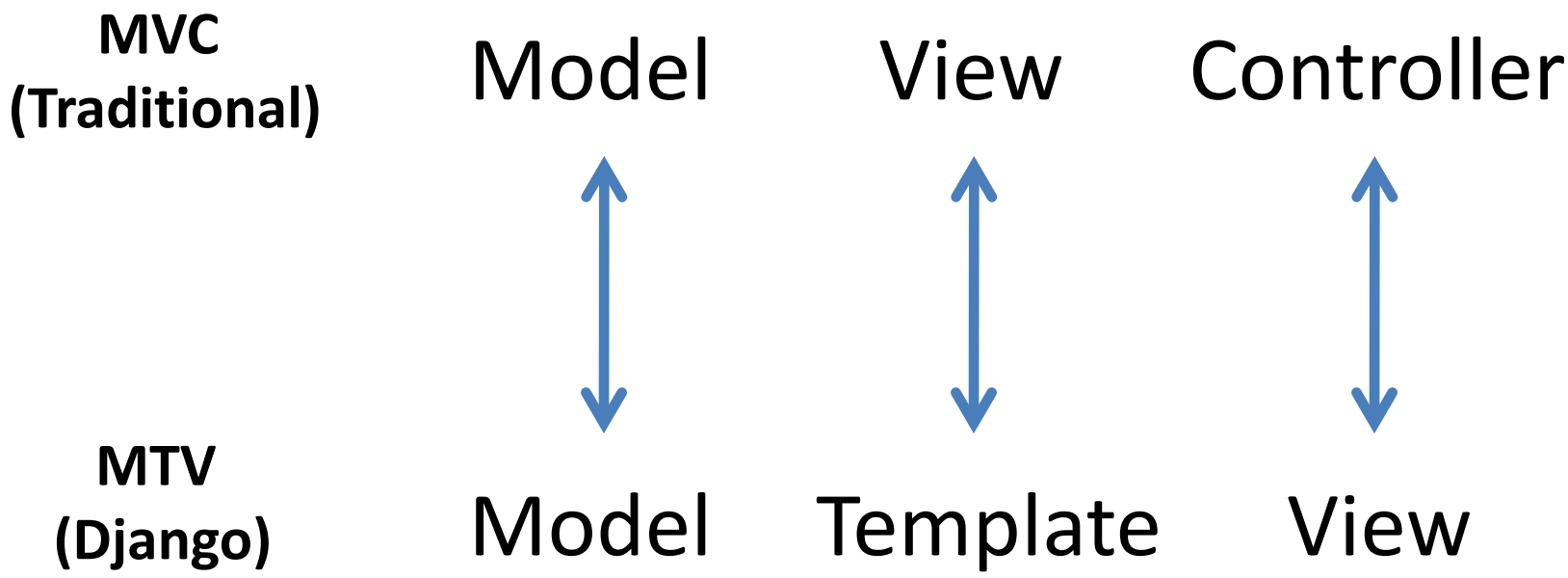
Accelerating Information Technology Innovation

Colombia Summer 2012
Lecture 3 – Django Views and Templates

MISTI

Google

Django Architecture



Django Views

Views

- Views are the logical interface between data (Models) and presentation (Templates)

Hello World

#inside views.py (create it)

```
from django.http import HttpResponse
    def hello(request):
        return HttpResponse("Hello world“)
```

EVERY view takes a request object as first parameter

EVERY view returns an HttpResponse object

How to hook it up?

```
#use urls.py

from django.conf.urls.defaults import *
from mysite.views import hello
        urlpatterns = patterns('',
        ('^hello/$', hello),
    )
```

Request Life Cycle

1. A request comes in to /hello/.
2. Django determines the root URLconf by looking at the ROOT_URLCONF setting.
3. Django looks at all of the URLpatterns in the URLconf for the first one that matches /hello/.
4. If it finds a match, it calls the associated view function.
5. The view function returns an HttpResponse.
6. Django converts the HttpResponse to the proper HTTP response, which results in a Web page.

A Note about Development

Where to start, views or URLconfs?

- Big Picture: Start with URLconfs
  - get an idea of what kind of content you need to deliver
  - to-do list
- Bottom Up: Start with Views
  - first make the pieces, then put the puzzle together

Generic Views

- Django comes with some commonly used views
  - redirect a user to another page
  - render a specific template
  - display list and detail view of objects
  - display date-based objects in archive pages

Generic Views

```
#Example: direct_to_template

from django.conf.urls.defaults import *
from django.views.generic.simple import direct_to_template

urlpatterns = patterns('',
    (r'^about/$', direct_to_template,  { 'template': 'about.html'
    })
)

#Magic!!
```

Loose Coupling

- Changes made to one piece of code should have little or no effect on other pieces of code
  - to change URL from “/hours_ahead” to “/plus_hours”, need to change only URLconf
  - to change View from calculating “hours ahead” to “hours ago”, need to change only view
  - Allows linking multiple URLs to the same view

Loose Coupling

```
def hours_ahead(request, offset):
    try: offset = int(offset)
    except ValueError: raise Http404()
    dt = datetime.datetime.now() +
          datetime.timedelta(hours=offset)
    html = "<html><body>In %s hour(s), it will be
    %s.</body></html>" % (offset, dt)
    return HttpResponse(html)

#HTML should be in a Template!!
```

Django Templates

Templates

- A text-based template for HTML, CSS, XML, JavaScript, etc.
- Mixture between hard-coded text and abstractions
- Abstractions
  - Variables
  - Tags
- Re-useable and extensible

weather.html

```
<html>
      <head>
        <title> Weather </title>
      </head>
      <body>
        <p>Today’s weather in {{ city }} is {{ description }}.</p>
        <div id=“temperature”>
        {% for day in thisWeek %}
                        <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
        {% endfor %}

        </div>
        <div id="ads">
        {% block ads %}
                Click on these ads!
        {% endblock %}
        </div>
      </body>
</html>
```

Context

```
city = ‘Cali‘
description = 'sunny‘
thisWeek = [dict(date='Thursday', temperature=20),
            dict(date='Friday', temperature=25),
            dict(date='Saturday', temperature=22)]
```

Displayed by browser

Today’s weather in Cali is sunny.

- On Thursday, the temperature will be 20.
- On Friday, the temperature will be 25.
- On Saturday, the temperature will be 22.

Click on these ads!

Syntax

*template.*render(*context)*

```
week = [dict(date='Thursday', temperature=20),
                dict(date='Friday', temperature=25),
                dict(date='Saturday', temperature=22)]

weather.render({city:‘Cali’, description:’sunny’,
        thisWeek=week})
```

Hard-coded Text in weather.html

```
<html>
    <head>
      <title> Weather </title>
    </head>
    <body>
      <p>Today's weather in {{ city }} is {{ description }}.</p>
      <div id="temperature">
      {% for day in thisWeek %}
                  <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
      {% endfor %}

      </div>
      <div id="ads">
      {% block ads %}
            Click on these ads!
      {% endblock %}
      </div>
    </body>
</html>
```

Variables

- `{{ variable }}`
  - If variable doesn't exist, then output `TEMPLATE_STRING_IF_INVALID` (default: empty string "")
- `{{ variable.attribute }}`
  1. Dictionary Lookup. `variable["attribute"]`
  2. Attribute Lookup. `variable.attribute`
  3. Method Call. `variable.attribute()`
  4. List-index Call. `variable[attribute]`

Variables in weather.html

```
<html>
    <head>
      <title> Weather </title>
    </head>
    <body>
      <p>Today's weather in {{ city }} is {{ description }}.</p>
      <div id="temperature"
      {% for day in thisWeek %}
                  <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
      {% endfor %}

      </div>
      <div id="ads">
      {% block ads %}
            Click on these ads!
      {% endblock %}
      </div>
    </body>
</html>
```

Filters

- Modify the output of variables
- `{{ variable|filter }}`

```
foo := “Hello World”
bar := [‘a’, ‘b’, ‘c’]

{{ foo|lower }} --> hello world
{{ bar|length }} --> 3
{{ bar|slice:“:2” }} --> [‘a’, ‘b’]
{{ some|default:“error!” }} --> error!
```

Tags

- for loops
- if clauses
- comments
- blocks
- and many more built-in tags (look them up!)

- {% tag %} … {% endtag %}

Tags in weather.html

```
<html>
    <head>
      <title> Weather </title>
    </head>
    <body>
      <p>Today's weather in {{ city }} is {{ description }}.</p>
      <div id="temperature"
      {% for day in thisWeek %}
                  <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
      {% endfor %}

      </div>
      <div id="ads">
      {% block ads %}
            Click on these ads!
      {% endblock %}
      </div>
    </body>
</html>
```

For loops

```
{% for x in y %}
    … logic …
{% endfor %}

fruit_basket := {‘apples’, ‘oranges’, ‘pineapples’}

{% for fruit in fruit_basket %}
  <li>{{ fruit }}</li>
{% endfor}


        <li>apples</li>
 -->    <li>orange</li>
        <li>pineapples</li>
```

If clauses

{% if <condition> %}

... logic ...

{% else %}

... logic ...

{% endif %}

```
{% if rain > 1 }
     Buy an umbrella for {{ price1 }}
{% else %}
     Buy sunglasses for {{ price2 }}
{% endif %}
```

Comments

```
{% comment %}
    This comment won't be displayed!
{% endcomment}
```

- Ignore everything inside tag
  - For inline comments, use {# blah blah blah #}

Template Inheritance

- Define extensible parts of a template with block tags

```
{% block name %}
 …
{% endblock %}
```

- Create child templates that can extend blocks
- Load parent template with

```
{% extends “parent_template” %}
```

weather.html

```
<html>
    <head>
      <title> Weather </title>
    </head>
    <body>
      <p>Today's weather in {{ city }} is {{ description }}.</p>
      <div id="temperature">
      {% for day in thisWeek %}
                <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
      {% endfor %}

      </div>
      <div id="ads">
      {% block ads %}
            Click on these ads!
      {% endblock %}
      </div>
    </body>
</html>
```

ads.html

```
{% extends "weather.html" %}
{% block ads %}
{% if rain > 1 }
    Buy an umbrella!
{% else %}
    Buy sunglasses!
{% endif %}
{% endblock %}
```

Context

```
city = ‘Cali‘
description = 'sunny‘
thisWeek = [dict(date='Thursday',temperature=20),
                dict(date='Friday', temperature=25),
                dict(date='Saturday', temperature=22)]
rain = 3
```

Displayed by browser

Today’s weather in Cali is sunny.

- On Thursday, the temperature will be 20.
- On Friday, the temperature will be 25.
- On Saturday, the temperature will be 22.

Click on these ads!

Buy an umbrella!

Template Inheritance

- In child template, redefine contents of the parent's block tag
  - similar to overriding methods in class inheritance
- If a block tag is not redefined, then use contents of block tag in parent
- {{ block.super }} explicitly refers to contents of block tag in parent

Templates

- Mixture of hard-coded text and abstractions
- Abstractions often look like and function like Python code, but you can’t run arbitrary Python code
  - Lookup list of built-in filters and tags in Django
  - Customize your own filters and tags
- Complex logic with arbitrary Python should be performed by views.py and only the processed variables should be passed to a template

Templates

Remember to specify where your templates are in TEMPLATE_DIRS in settings.py