



# Server-Side Languages

Todd Smith  
[tbsmith@fullsail.com](mailto:tbsmith@fullsail.com)

## Welcome to SSL Day 7!

Intro to Frameworks

Day 7



# Server-Side Languages

## PHP Framework: CodeIgniter



clone this:

<https://github.com/EllisLab/CodeIgniter.git>

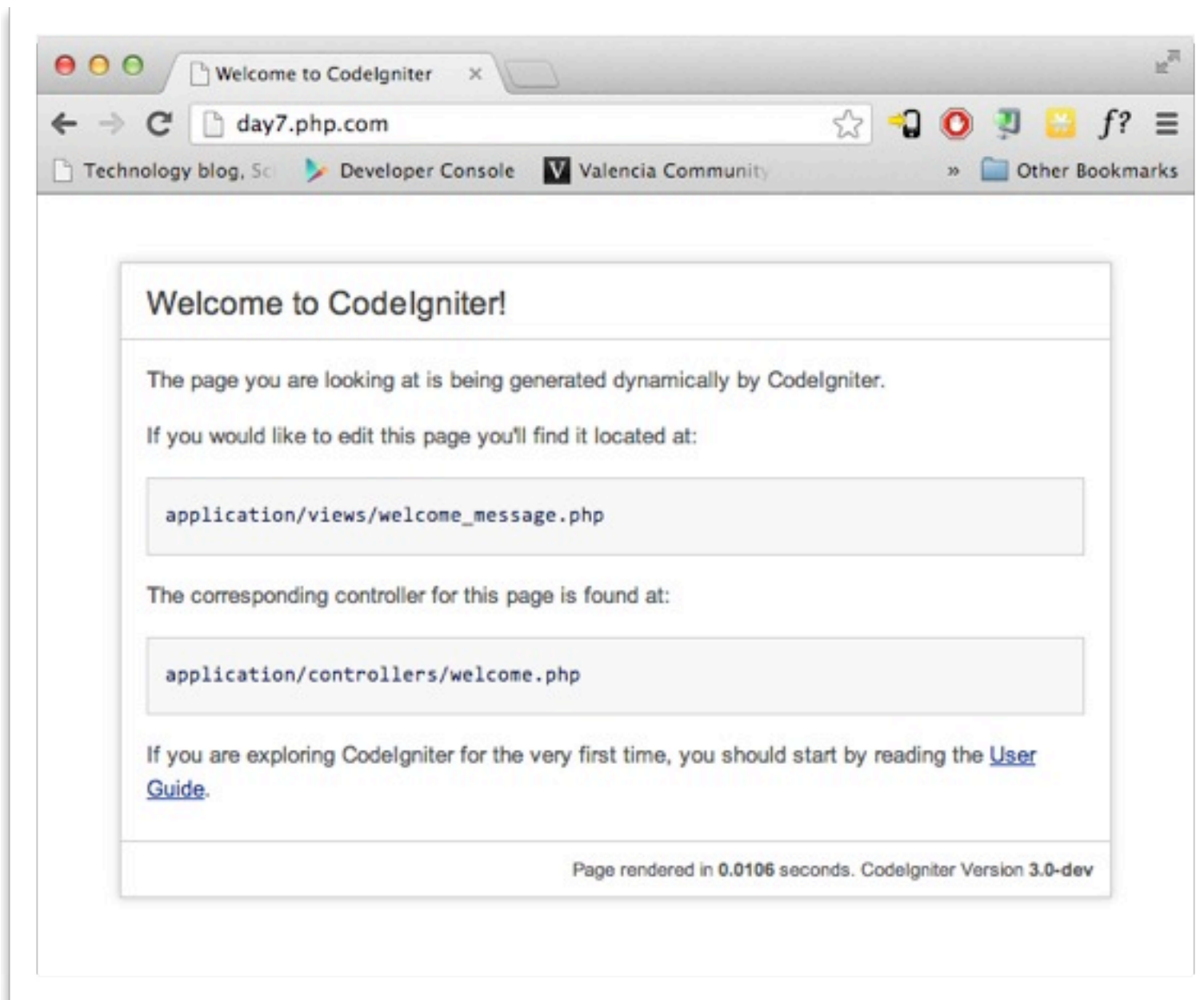
The files in the CodeIgniter folder  
will become your new web root

Day 7



# Server-Side Languages

Test the site





# Server-Side Languages

## Class Assessment

Search the web for “codeigniter controllers”

Follow the first 5 sections of this tutorial.

### CodeIgniter User Guide Version 2.1.3

---

[CodeIgniter Home](#) > [User Guide Home](#) > [Controllers](#)

## Controllers

---

Controllers are the heart of your application, as they

- ▣ [What is a Controller?](#)
- ▣ [Hello World](#)
- ▣ [Functions](#)
- ▣ [Passing URI Segments to Your Functions](#)
- ▣ [Defining a Default Controller](#)

Day 7



# Server-Side Languages

## Python Framework: Django



Check  
FSO for  
the license  
number!

download this:

<http://www.jetbrains.com/pycharm/download/index.html>

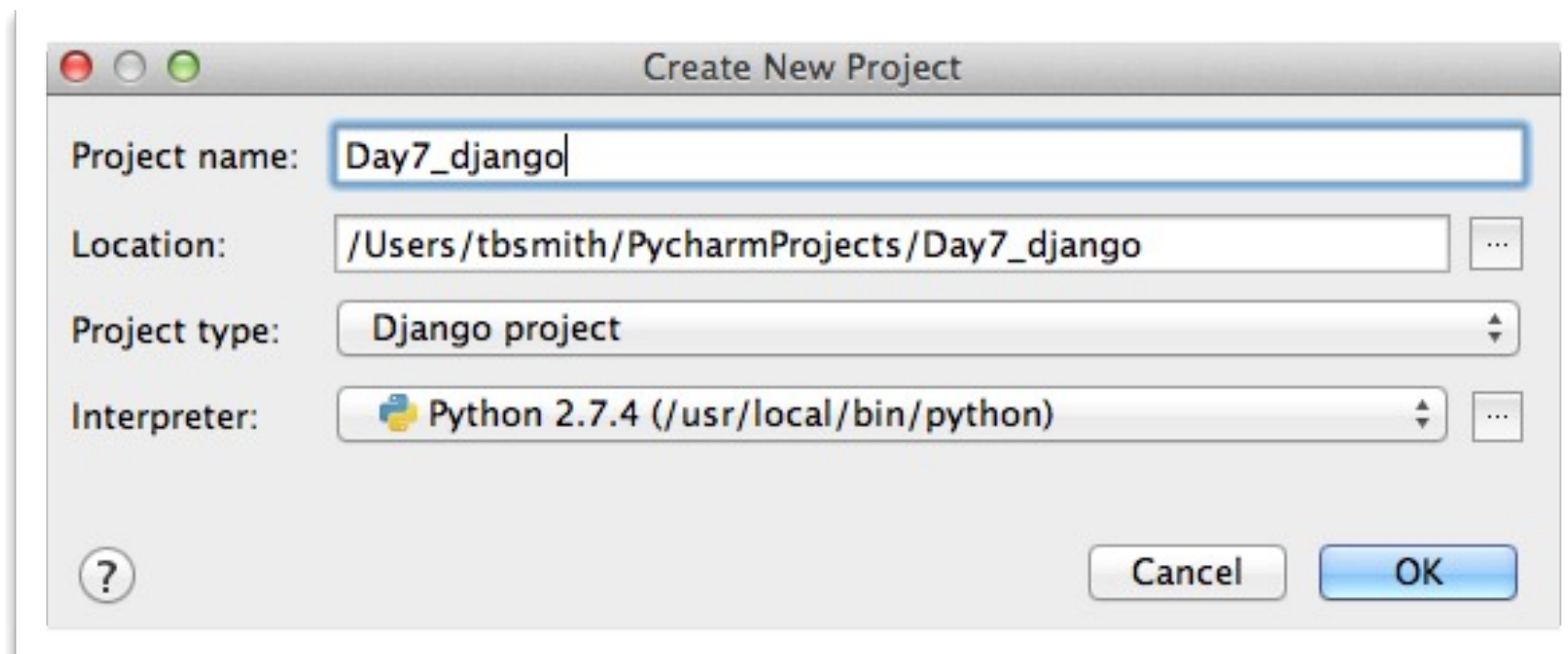
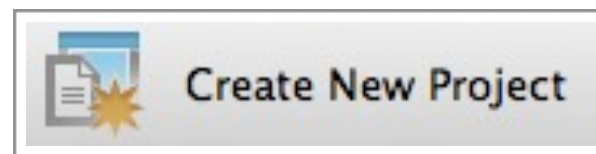
The PyCharm IDE will handle our directory structure

Day 7



# Server-Side Languages

## Starting the PyCharm Project



Day 7





# Server-Side Languages

## Starting the PyCharm Project

You may set up different apps for a website's blog, JSON API, admin section, shopping cart, analytics, user accounts...

The idea is to keep your code modular and reusable.

Django Project Settings

Project name: Day7\_django

Application name: website

Templates folder: /Users/tbsmith/PycharmProjects/Day7\_django/templates

☒ Enable Django admin

Cancel OK

Django provides an already-built admin section to maintain everything in your database and in particular user accounts.

Day 7



# Server-Side Languages

## Set up your urls

A regular expression goes between the ticks

```
from django.conf.urls import patterns, url

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^$', 'website.views.home', name='home'),
    url(r'^home$', 'website.views.home', name='home'),
    url(r'^about$', 'website.views.about', name='about'),

    url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
    url(r'^admin/', include(admin.site.urls)),
)
```

Now django will be expecting us to have a home class in the views.py file in the website app

Day 7





# Server-Side Languages

## Set up your views

The screenshot shows a PyCharm IDE window. On the left, the 'Project' tool window displays the file structure of a Django project named 'Day7\_django'. The structure includes a 'Day7\_django' folder with files like '\_\_init\_\_.py', 'settings.py', 'urls.py', and 'wsgi.py', a 'templates' folder, and a 'website' folder containing '\_\_init\_\_.py', 'models.py', 'tests.py', and 'views.py'. The 'views.py' file is selected and highlighted. On the right, the 'views.py' file is open in the editor, showing the following Python code:

```
from django.http import HttpResponse

def home(request):
    html = "<html><body>My Home Page</body></html>"
    return HttpResponse(html)

def about(request):
    html = "<html><body>About Us</body></html>"
    return HttpResponse(html)
```

Below the code editor, there is a callout box titled 'Request and response objects' which contains information about the `HttpRequest.method` attribute.

**HttpRequest.method**

A string representing the HTTP method used in the request. This is guaranteed to be uppercase. Example:

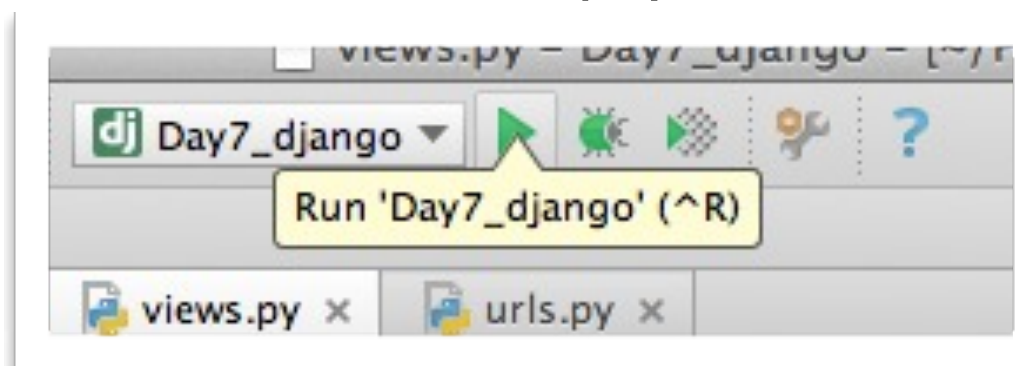
```
if request.method == 'GET':
    do_something()
elif request.method == 'POST':
    do_something_else()
```



# Server-Side Languages

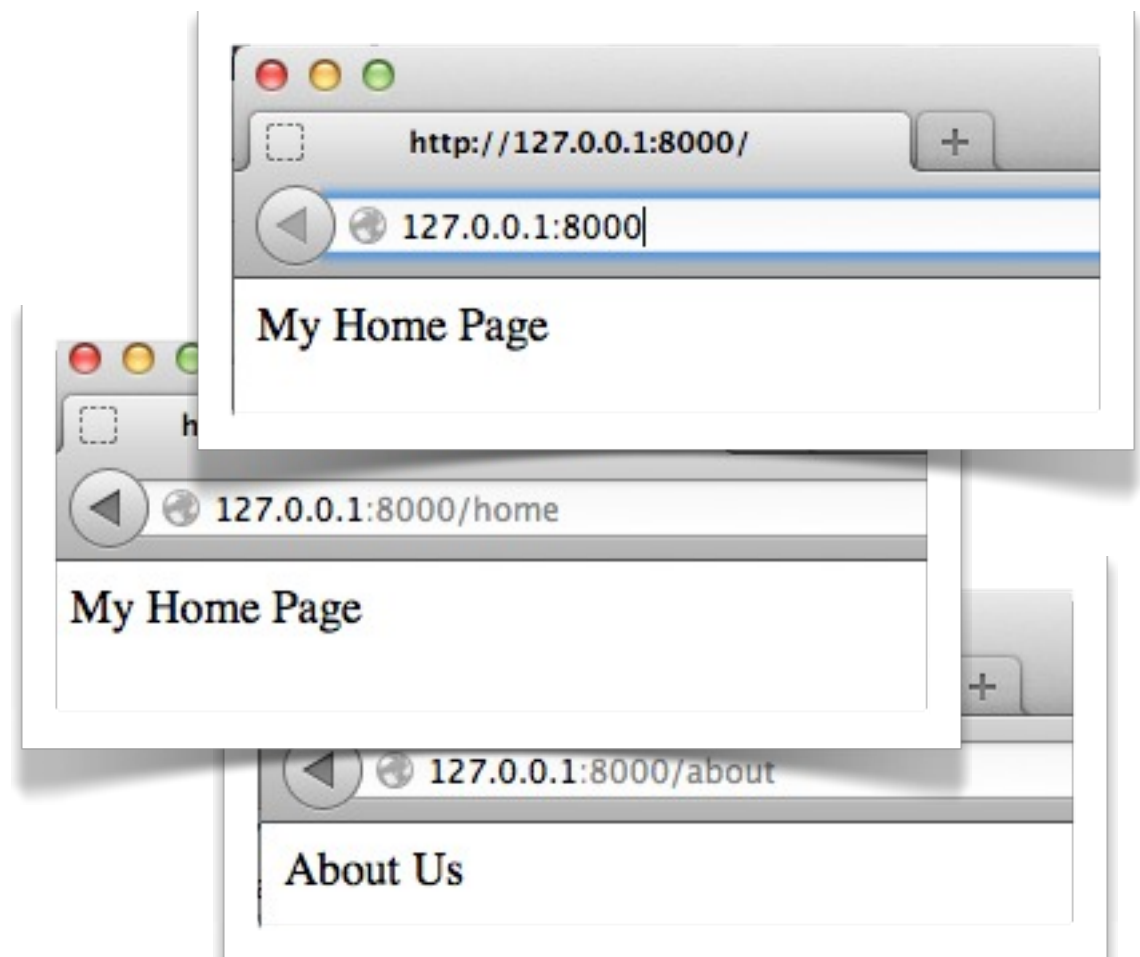
## Test your views

Click play



Click the link

```
5:44  
, using settings 'Day7_django.settings'  
s running at http://127.0.0.1:8000/  
CONTROL-C.  
dress already in use
```



Day 7



# Server-Side Languages

## The Templating System

```
from django.http import HttpResponse
from django.template.loader import render_to_string

def home(request):
    data = {}
    data['title'] = "Home"
    data['user_message'] = "You're in home"

    return HttpResponse (
        render_to_string('header.html', data) +\
        render_to_string('home.html', data) +\
        render_to_string('footer.html', data)
    )

def about(request):
    data = {}
    data['title'] = "About"
    data['user_message'] = "You're in about"
    return HttpResponse (
        render_to_string('header.html', data) +\
        render_to_string('about.html', data) +\
        render_to_string('footer.html', data)
    )
```

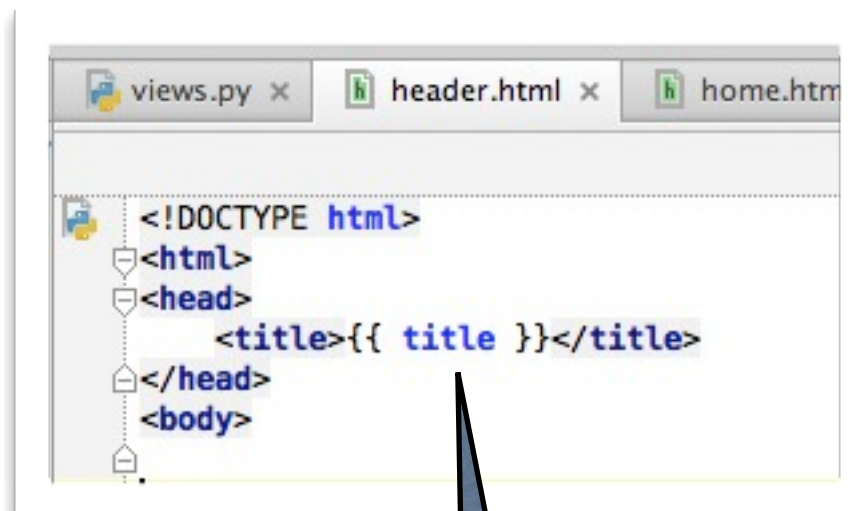
Day 7





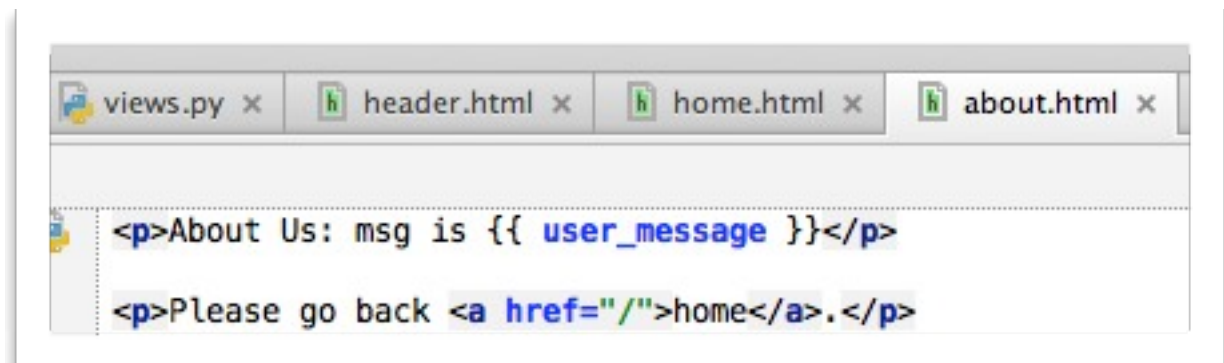
# Server-Side Languages

## The Templating System

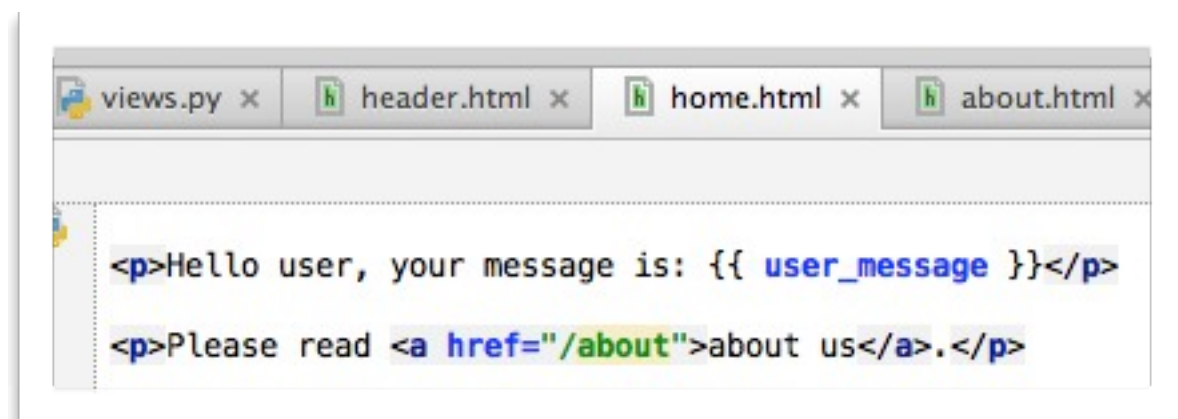


```
views.py x header.html x home.htm
<!DOCTYPE html>
<html>
<head>
  <title>{{ title }}</title>
</head>
<body>
```

These are the keys  
in the dictionary that was  
passed



```
views.py x header.html x home.html x about.html x
<p>About Us: msg is {{ user_message }}</p>
<p>Please go back <a href="/">home</a>.</p>
```



```
views.py x header.html x home.html x about.html x
<p>Hello user, your message is: {{ user_message }}</p>
<p>Please read <a href="/about">about us</a>.</p>
```

[Link to docs](#)

Day 7

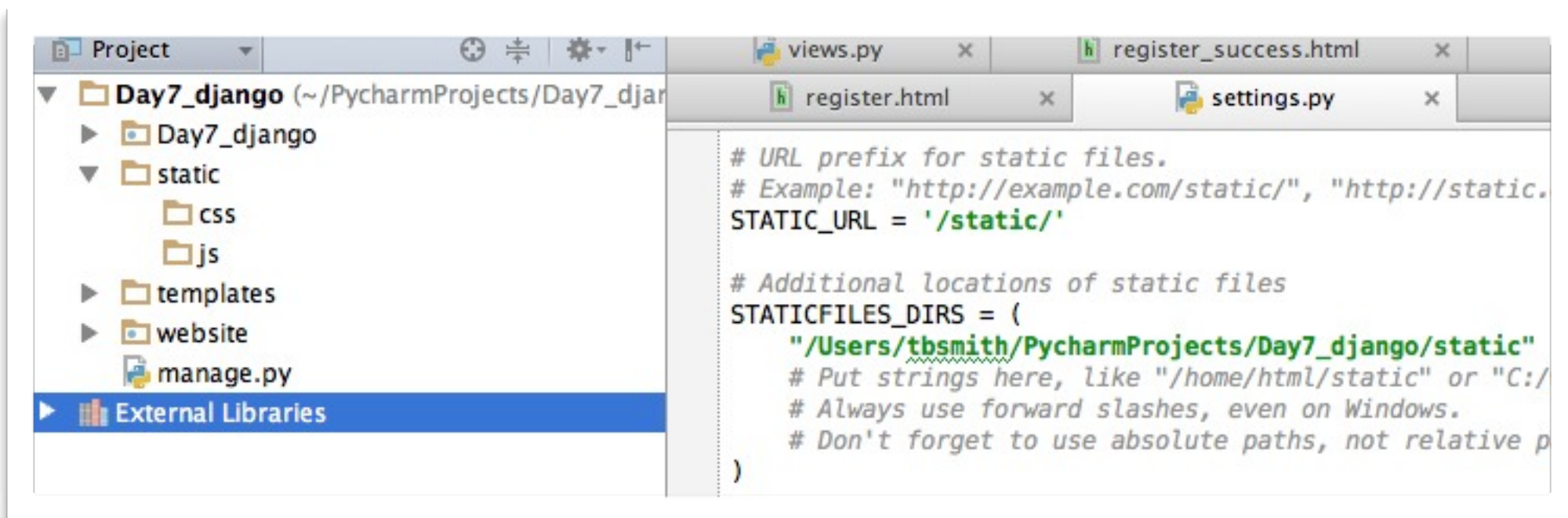


# Server-Side Languages

## Static Files

[docs](#)

Set your static url and directories  
in the settings.py file



Day 7





# Server-Side Languages

## Static Files

[docs](#)

Use static template tags

```
<!DOCTYPE html>
<html>
<head>
  {% load staticfiles %}

  <link type="text/css" rel="stylesheet" href="{% static 'css/foundation.css' %}" />
  <link type="text/css" rel="stylesheet" href="{% static 'css/normalize.css' %}" />

  <script src="{% static 'js/vendor/custom.modernizr.js' %}"></script>

  <title>{{ title }}</title>
</head>
<body>
```

Day 7



# Server-Side Languages

## Static Files

Run the `collectstatic` task in `manage.py`

Day 7



# Server-Side Languages

## A Simple Form: templates

```
<form action="/register" method="POST">
    {% csrf_token %}
    Username: <input name="username"/><br/>
    <input type="submit"/>
</form>
```

{% csrf\_token %} will prevent cross-site request forgeries. It is required for every form with method POST.

```
<p>You have registered with username {{ username }}.</p>
```

Day 7



# Server-Side Languages

## A Simple Form: views

```
from django.template import RequestContext
```

```
def register(request):
    if request.method == 'GET':
        return HttpResponse (
            render_to_string('header.html') +\
            render_to_string('register.html', RequestContext(request)) +\
            render_to_string('footer.html')
        )
    elif request.method == 'POST':

        # Validate form
        # Add user to database

        data = {}
        data['username'] = request.POST['username']

        return HttpResponse (
            render_to_string('header.html') +\
            render_to_string('register_success.html', data) +\
            render_to_string('footer.html')
        )
```

This should be passed to the templates with forms with method POST.

The + is for string concatenation, the \ is to let me continue the code on the next line.





# Server-Side Languages

## A Simple Form: testing

The image displays three overlapping browser window screenshots illustrating the registration process:

- Top-left window:** Shows the initial registration form at `127.0.0.1:8000/register`. It contains a "Username:" label, an empty text input field, and a "Submit Query" button.
- Middle window:** Shows the form after the user has entered "John" in the username field. The "Submit Query" button is still visible.
- Bottom-right window:** Shows the result of the registration, displaying the message: "You have registered with username John."





# Server-Side Languages

## Class Review

You should be able to make a barebones website in both CodeIgniter and Django.

Day 7



# Server-Side Languages

## Lab 7

Do this Django tutorial:

<http://lightbird.net/dbpedia/blog.html>

Day 7