

Slide 1:

What Django is

Python-based open-source web application framework written by Adrian Holovaty, Simon Willison, Wilson Miner and Jacob Kaplan Moss. It was initially released in July 05 and has picked a lot of steam since then... in fact 1.0 just went alpha a couple of weeks ago. Django was created when these guys started using Python as a replacement for PHP and Perl in their newsroom environment. As a result, it excels at delivering media and content rich applications.

Like a lot of different frameworks, Django has all the fixings **Limits repetitive work and gets apps up and running quickly... I'm going to show you a lot of those features.**

Finally Django does a great job Encourages best practices for building dynamic web applications: DRY, Separation of concerns, Restful URLs, good stuff that leads to great encapsulation and reusable code.

What Django isn't

A CMS – definitely not a CMS. Not full featured enough (though people confuse it because of the admin UI). Great platform to build a CMS on though (ideal) and there exists a lot of open source projects available that would make assembling a CMS easy out of Django...

A replacement for other web frameworks – Every web framework has its fans and its features... I like Django because it suits the sorts of things we build. Came across it around March of this year after being frustrated with a couple of Joomla, Wordpress deployments we did and playing with Rails but thinking it wasn't necessarily well suited to our problems. Django is the perfect platform for our company given our focus on branding, marketing, UX and content.

Slide 2

Object-Relation-Mapper

Define your data models as Python classes. The classes are used to generate DB tables and create and manipulate rows in the database.

Support for Postgres, mysql, oracle db's

Automatic Admin Interface

Many people consider this the killer feature. Takes the models and creates a great, highly usable admin interface automatically for you

Flexible Elegant URL mapping

RegEx based URL mapping that's easy to use and not framework dependent.

Template System

Straight forward, as close as you can get to HTML, templating system. Easy for designers and non-coders to understand. Easily extensible.

Cache System

done via memcached

high performance and straightforward to deploy

caching can be as granular as you like

Internationalization

generally a give-in with modern frameworks. Easily create language tables, etc... to do internationalization.

Slide 3 – Start Project

- 1)run django-admin to start a **site** – django-admin.py startproject
- 2)List directory, show people the contents – explain manage.py
- 3)Start an application – Separation of concerns and encapsulation - python manage.py startapp pr
- 4)Start textmate – Next Step

Slide 4 – Define Models, Sync DB

Note that the application we created has two stub files, models and views

Make the models

We have

Stories

Photos

Add app to settings, sync db

Slide 5, Admin – URLs, Models 1, Settings, Sync, Models 2

Fun stuff now... before we try running the app for the first time, we're going to need a way to interact with the data. We'll do that by using the admin interface

Slide 6, Public Views – URLs, View Functions

Slide 7, Templates – Views, Settings (switch DB), Template Dir, Templates

Now you're starting to see how Django apps are structured. Now lets use the templating system to make our views more useful and pretty.

Explain context passing

first lets tell our views to render to templates instead

Slide 8, Forms – New Model, Sync DB, Make a Form, Edit View, Edit Template

Finally, we're going to add a way for people to request information on our PR pieces

Slide 9

Language

6 of 1, half dozen the other.
IMHO, Python and Ruby are equivalent languages
Lets not start holy wars

Project Accelerators

Django wins
Generally you'll throw out the CRUD and views while the Admin interface is production quality
The AJAX and Javascript helpers are nice but has a tendency to tie you to one interaction model and it's hard to use JS libraries other than the Prototype/Scriptaculous combo

Template system

Django templates are more narrowly defined than Ruby's. Easier for non-programmers but potentially frustrating to more advanced users. Draw

ORM

Django defines its models and the fields on its models as a first order Python classes (similar to Hibernate in Java). Rails uses introspection on the database via active record and naming conventions on columns. IMO, Django's ORM is a little easier to grok but harder to evolve than the Rails version. Draw.

Conclusion:

- Use Rails in highly iterative, more application based projects. Use Django for more media/content driven apps where you have some idea of the models.

Final Slide

Interesting Projects

-Good separation of concerns means there's a lot of code out there for you to use and easily integrate into your projects

Hosting and Deployment

-Can get Django running in pretty much any environment that runs Apache, you have mod_python and the ability to modify the python path and add new python modules
-some have higher and lower degrees of difficulty (even on 1&1)
-recommend WebFaction, MediaTemple in beta

Further Reading - lots of books about to come out because of 1.0 and Django