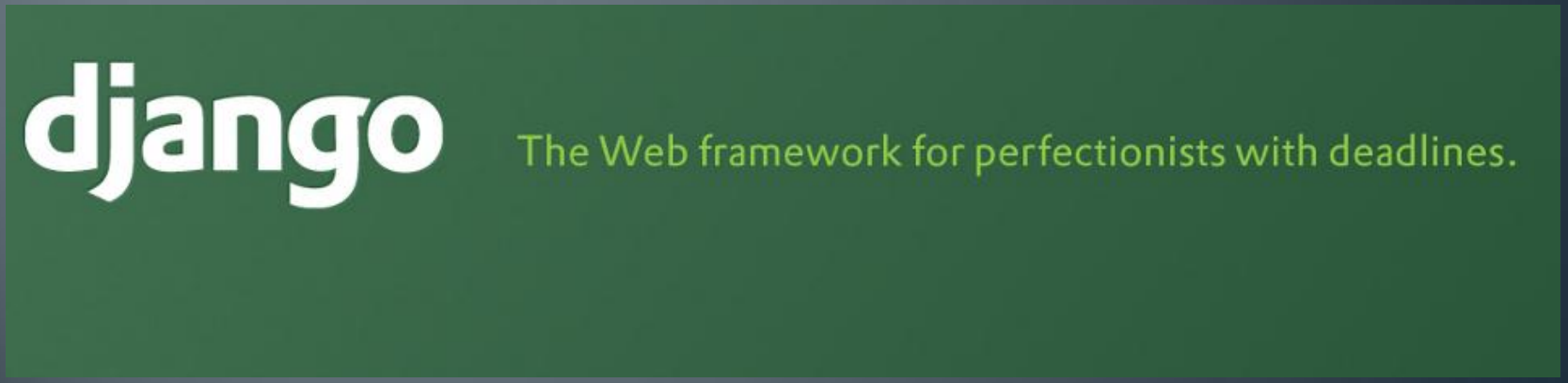
Django

-Abhijit Mahajan

Introduction

- **Django web Framework**
  - is an open source Web 2.0 application framework
  - written in Python
  - which follows the model-view-controller architectural pattern.



History

It was originally developed to manage several news-oriented sites for The World Company of Lawrence, Kansas, and was released publicly under a BSD license in July 2005; the framework was named after guitarist Django Reinhardt. In June 2008 it was announced that a newly formed Django Software Foundation will maintain Django in the future.

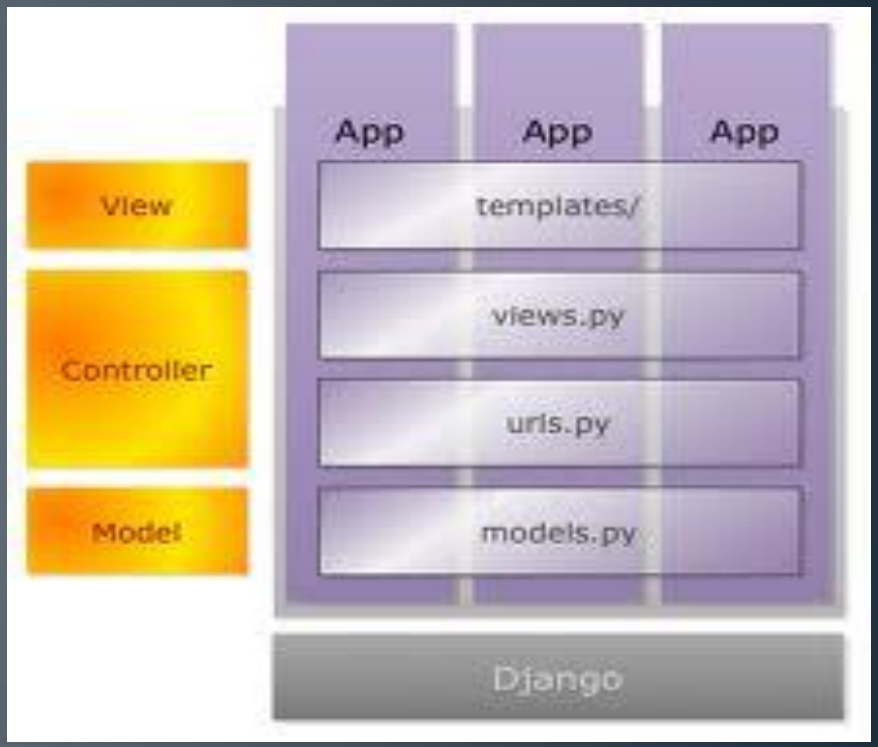
Highlights

- Django's primary goal:
  - ease the creation of complex, database-driven websites.
  - emphasizes reusability and pluggability of components
  - rapid development
  - and the principle of ”don't repeat yourself”
- Python is used throughout
  - for settings, files, and data models.
- Django provides an optional administrative interface that is generated dynamically through introspection and configured via admin models.

Components

The main Django **MVC** framework
Is made up of an object-relational
mapper which sits between
data models (defined as Python
classes) and

- a relational database ("**M**odel")
  which processes requests with
- a web templating system ("**V**iew")
- and a regex-based
  URL dispatcher ("**C**ontroller").



Django MVC

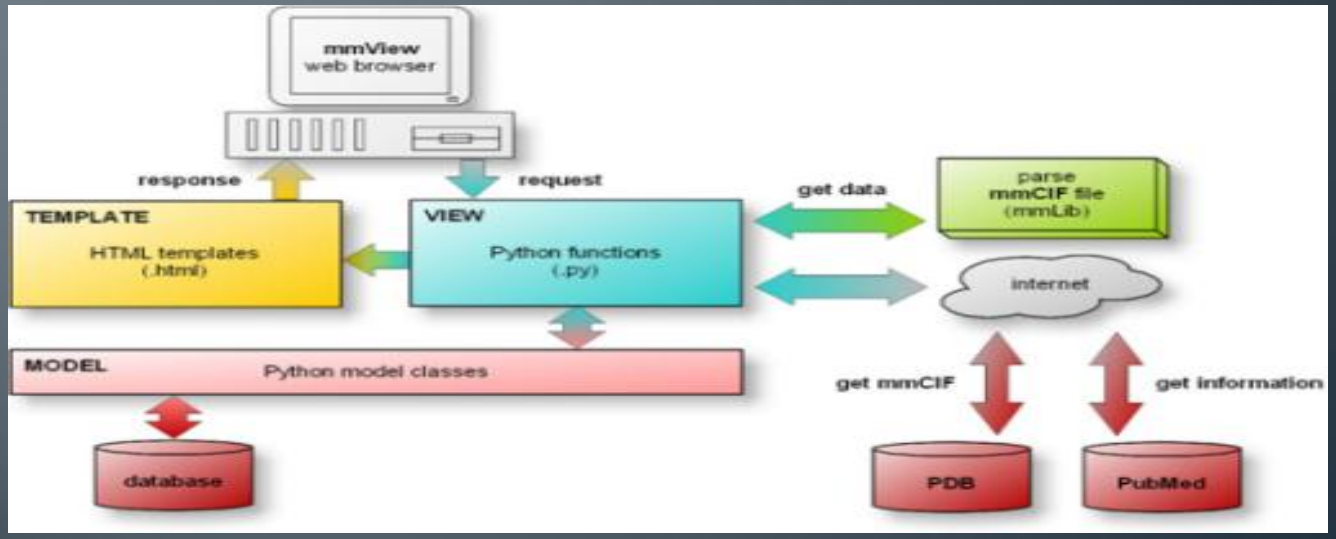
- Django follows this MVC pattern closely enough. This is how M, V, and C work together in Django:
- "M":
  - is the data-access part
  - handled by Django's database layer
- "V":
  - is the one that selects which data to display and how to do it
  - handled by views and templates.
- "C":
  - This part delegates to a view depending on user input
  - handled by the framework itself
  - follows URLconf and calls the appropriate Python function for the given URL (explained in the later slides).

Django "MTV"

- Since "Controller" is handled by the framework and most of the things in Django happens in models, templates and views, Django has been referred to as an *MTV framework* where,
- *"Model"*:
  - Is the data access layer.
  - contains all details about accessing/validating the data, behavior of data, and their relationships.
- "Template":
  - Is the presentation layer.
  - decides how something should be displayed on a Web page or other type of document.

Django "MTV"...

- "View":
  - Is the business logic layer.
  - has the logic that access the model and gives to the appropriate template(s).
  - it is a bridge between models and templates.



Django Features

Also included in the core framework are:

- Web server
  - It is lightweight and standalone
- A form serialization and validation system
  - translates between HTML forms and values suitable for storage in the database.
- A caching framework
  - any of several cache methods can be used
- Support for middleware classes
  - Help is providing custom functionalities

Django Features

- An internal dispatcher system
  - Facilitates component communication via pre-defined signals.
- An internationalization system
  - Has translations of Django's own components into a variety of languages.
- A serialization system
  - produces and reads XML and/or JSON representations of Django model objects.
- A system for extending the capabilities of the template engine.
- An interface
  - to Python's built-in unit test framework.

Bundled applications

The main Django distribution also bundles a number of applications in its "contrib" package, including:

- An authentication system.
- The dynamic admin interface.
- RSS and Atom syndication feed generation tools.
- A flexible commenting system.

Bundled applications…

- A sites framework
  - allows us to run multiple websites, each with their own content and applications.
- Tools for generating Google Sitemaps.
- Tools for preventing cross-site request forgery.
- Template libraries
  - enable the use of lightweight markup languages (Textile and Markdown)
- A framework for creating GIS applications.

Support for db backend

- Django officially supports four database backends:
  - PostgreSQL
  - MySQL
  - SQLite
  - Oracle.
- Microsoft SQL Server can be used with django-mssql in Microsoft OS.
- External backends exist for IBM DB2, SQL Anywhere and Firebird.
- “django-nonrel” supports NoSQL databases, such as MongoDB and Google App Engine's Datastore.

- Django may also be run in conjunction with Jython on any Java EE application server such as GlassFish or Jboss
  - in this case django-jython must be installed which will provide JDBC drivers for database connectivity.
  - and also provides functionality to compile Django in to a .war suitable for deployment.

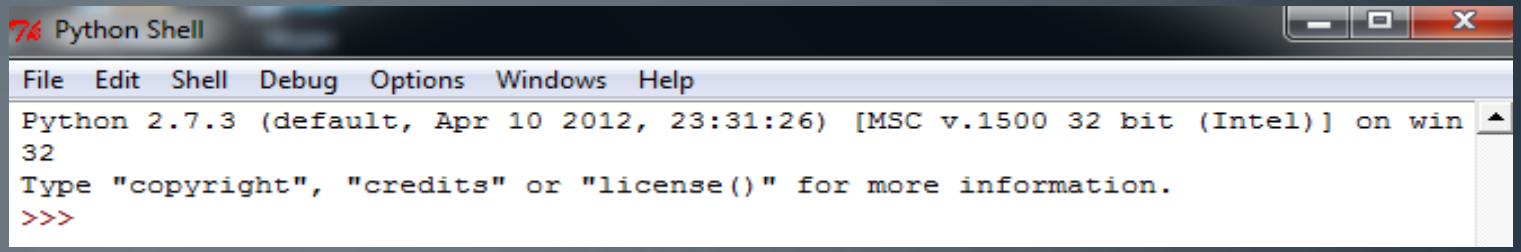
Development environments

- Text editors such as Vim, Emacs or TextMate with Django Bundle can be used
- Specialized tools providing debugging, refactoring, unit testing, etc can also be used like:
  - Komodo IDE
  - Eclipse with PyDev
  - PyCharm
  - NetBeans with Django Plugin
  - Wing IDE
  - Eric Python IDE
  - Microsoft Visual Studio with Python Tools for Visual Studio

Installation steps

Quick install guide

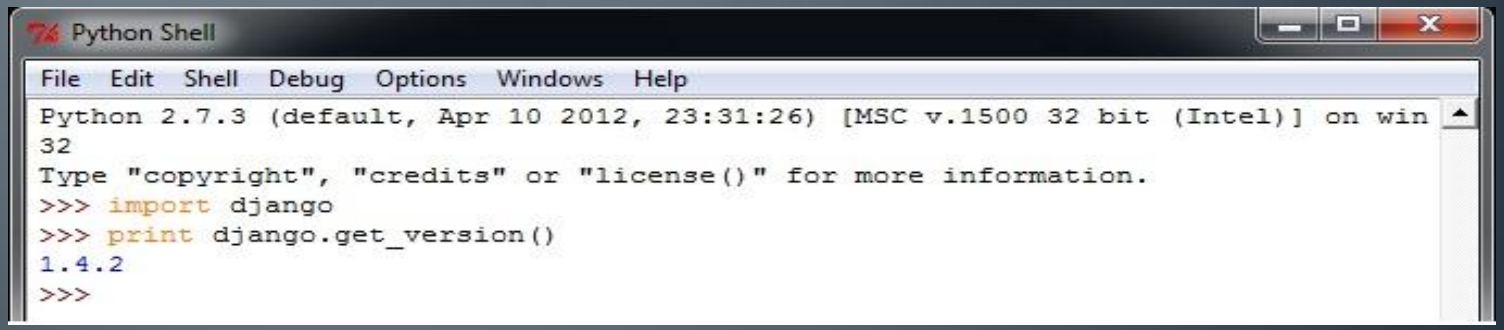
- Install Python – Any version from 2.5 to 2.7



- Set up a database – PostgreSQL, MySQL, Oracle, etc. SQLite is already installed in Python 2.5 or later.
- Remove any old versions of Django (if upgrading)

Installation steps…

- Install Django using any 1 of these options:
  - Install a version of Django provided by the OS.
  - Install an official release.
  - Install the latest development version.
- Verify



Steps to create a Django app

- Creating a project
  - Project is a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings
  - Run the command “django-admin.py startproject mysite”
  - A ”mysite” directory is created with the following 4 files:
    - __init__.py - An empty file that tells Python that this directory should be considered a Python package.
    - settings.py - Settings/configuration for this Django project.
    - urls.py - The URL declarations for this Django project; a "table of contents" of your Django-powered site.
    - wsgi.py - An entry-point for WSGI-compatible webservers to serve your project.

Creating a Project…

- Every website will have its own corresponding project.
- The development server
  - Django comes with a lightweight web-server written purely in Python
  - Run the command “python manage.py runserver” to start it.
- Database setup
  - Edit ”mysite/settings.py”
  - Change the ENGINE, NAME, USER, PASSWORD and HOST keys in the DATABASES 'default' item to match your database connection settings.

Apps

- INSTALLED_APPS setting holds the names of all Django applications activated in the Django instance.
- Execute “python manage.py startapp app-name” to create an app
- Apps can be used in multiple projects.
- They can be packaged and distributed for use by others in their projects.
- By default, INSTALLED_APPS contains few apps:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
)
```

- Every application uses at least one database table, so, create the tables in the database by running “python manage.py syncdb”.

Creating models

- A Django model is a description of the data in your database, represented as Python code.
- Django uses a model to execute SQL code in the background and return Python data structures representing the rows of the database tables.
- Relation between Django Model and Table in DB:
  - Each model generally corresponds to a single database table.
  - Each attribute on a model generally corresponds to a column in that database table.
  - The attribute name corresponds to the column's name
  - The type of field corresponds to the database column type
- Django gives an automatically-generated database-access API.

Creating models…

- Each model is represented by a class which is a subclass of django.db.models.Model.
- Class variables of a model represent a database field in the model.
- Each field is represented by an instance of a Field class -- e.g., CharField for character fields and DateTimeField for datetimes.
- The name of each Field instance is the field's name. This value is used in the Python code, and the database will use it as the column name.

Example of a Model

File Edit Format Run Options Windows Help

```
from django.db import models

class Musician(models.Model):
        first_name = models.CharField(max_length=20)
        last_name = models.CharField(max_length=20)
        instrument = models.CharField(max_length=30)
class Album(models.Model):
        artist = models.ForeignKey(Musician)
        name = models.CharField(max_length=30)
        release_date = models.DateField()
```

- As we can see in the class “Album”, Django allows the use of relations. Here the Musician is used as a foreign key in Album class.
- Objects can be used to access the models.

Activating models

- After defining models, Django has to be informed that they will be used.
- This is done by editing the settings file: add the name of the module that contains your models.py to the INSTALLED_APPS setting.
- Ex: if the models are in the module mysite.myapp.models, INSTALLED_APPS should have

```
INSTALLED_APPS = (
    #...
    'mysite.myapp',
    #... )
```

- After adding new apps to INSTALLED_APPS, run “manage.py syncdb”.

Views

- “View” is just a Python function that takes an HttpRequest as and returns an HttpResponse.

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello world")
```

- Views is nothing but a webpage
- Views retrieve data according to some parameters, load templates and render them (with the data that is retrieved).
- It’s a good practice to keep the View unaware of the template system being used.
- GET and POST should be easily differentiable by the view.

URLconf

- To hook a view function to a particular URL with Django, URLconf is used.
- URLconf is a mapping between the URLs and their corresponding view function.
- When “django-admin.py startproject” is executed, a URLconf is created automatically (the file urls.py).

```
from django.conf.urls.defaults import *
from mysite.views import hello

urlpatterns = patterns('',
    ('^hello/$', hello),
)
```

How Django handles Requests

- This is how URL's are interpreted by Django when a request comes in for , say, /news/
- Django determines root URLconf by looking at the ROOT_URLCONF in settings.py
- Django looks at all of the URLpatterns in the URLconf for the first one that matches /news/.
- If it finds a match, it calls the view function associated with the URL.
- Django converts the HttpResponse to the proper HTTP response and gives the webpage. The following is the urls.py file:
- The command "python manage.py runserver" is used to test to see if the url function's

Templates

- Templates are used to separate logic from the presentation
  - they are reusable because of this.
- Templates render the details given by the HTTPResponse
- Templates are strings which are merged with data to produce the output.
- Django templates contain place holders for information from the database.
- After the placeholders are substituted with values, the result is returned as HTML.

Sample Template

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

Variable

Logic Tag

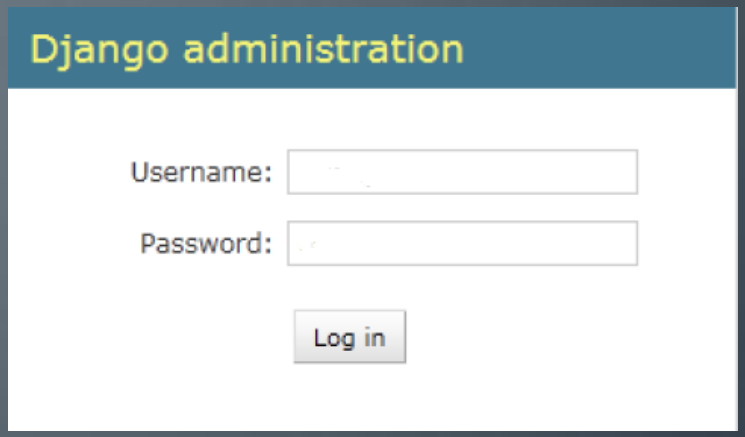
End of logic tag

Admin Site

- Since building the admin site is not fun, Django provides an automatic admin interface.
- This works by reading metadata in the model to provide a powerful and production-ready interface that is ready for immediate use by site admins.
- Since the Django admin site is optional, certain steps have to be taken.
  - First, Make the following canges to settings file:
    - Add 'django.contrib.admin' to the INSTALLED_APPS setting.
    - Make sure INSTALLED_APPS contains 'django.contrib.auth', 'django.contrib.contenttypes‘ and 'django.contrib.sessions'.
    - Make sure MIDDLEWARE_CLASSES contains 'django.middleware.common.CommonMiddleware', 'django.contrib.sessions.middleware.SessionMiddleware‘ and 'django.contrib.auth.middleware.AuthenticationMiddleware'.

Admin site…

- Second, run python manage.py syncdb. This step will install the extra database tables that the admin interface uses.
- Third, add the admin site to your URLconf (in urls.py, remember).
  - "urls.py" generated by "django-admin.py startproject" contains commented-out code for the Django admin, just uncomment it.
- Run the development server and visit http://127.0.0.1:8000/admin/ in the Web browser.



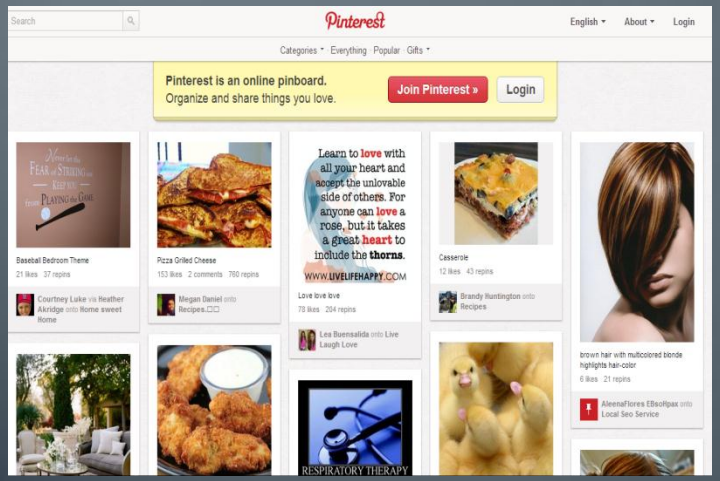
DjangoCon

- There is a twice-yearly conference for Django developers and users, named "DjangoCon"
- It has been held since September 2008.
- One DjangoCon a year is held in Europe, in May or June;
- The other is held in the United States in September, usually in Portland, Oregon.
- The 2012 DjangoCon took place in Washington D.C from 3 to 8 September.

Django in-use

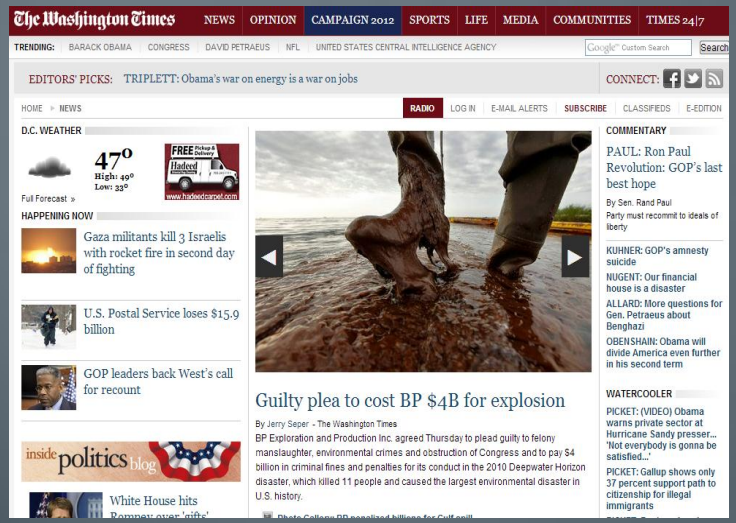
Some well known sites that use Django include:

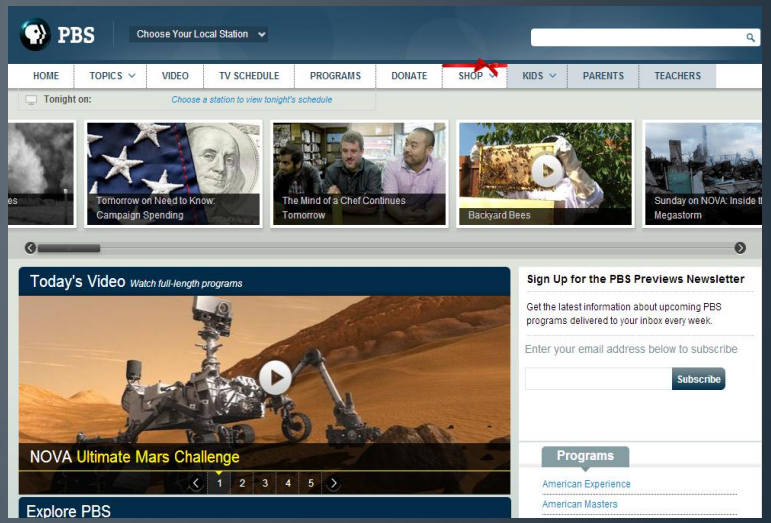
- Pinterest
- Instagram



Django in-use…

- The Washington Times
- Public Broadcasting Service





References

- Wikipedia
- www.djangoproject.com – This site is the best place to start. Has a lot of material to understand things better.
- www.djangobook.com – Has explanation about everything in detail.
- Google Images – Used it for a few images in this presentation.