Django

The Web Framework for perfectionists with deadlines

Presented by: Soumya Sundaram

Overview

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

– www.djangoproject.com

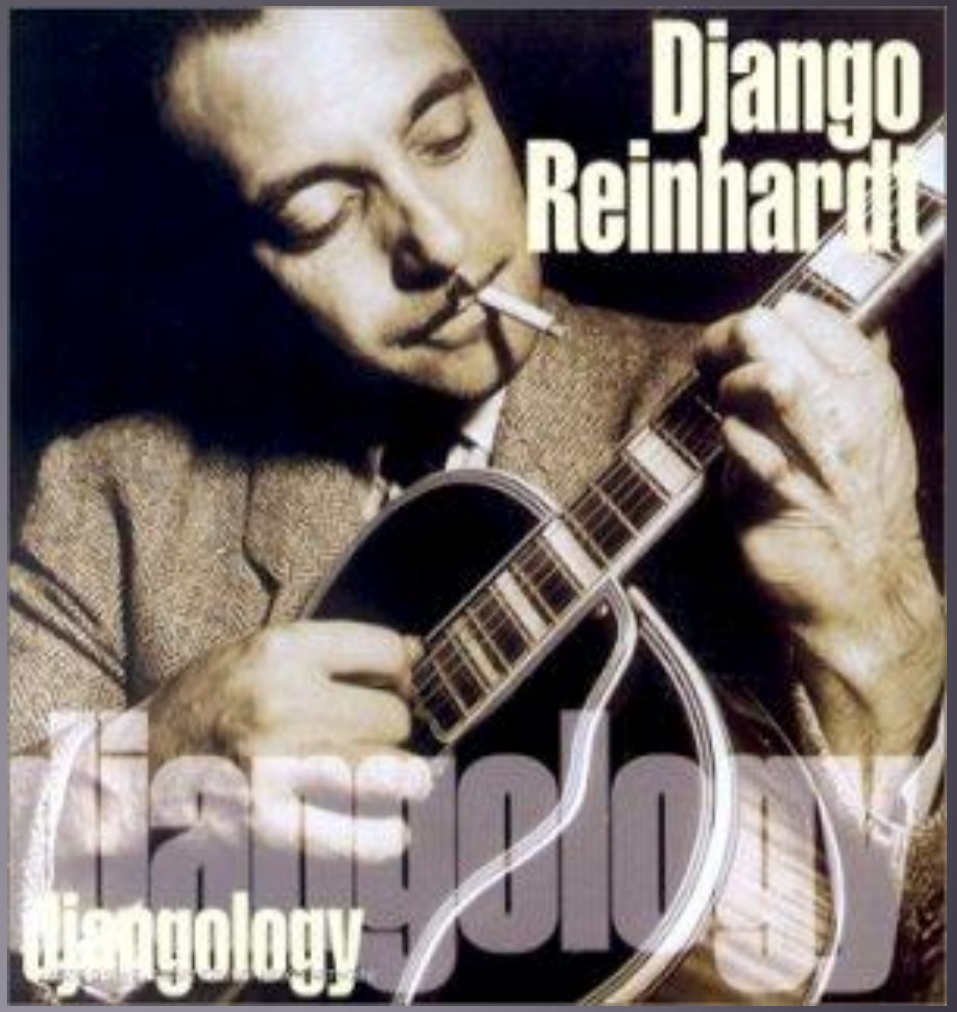
- Django is a part of the “third generation” Web framework after HTML and PHP
- Django focuses on building dynamic and database driven sites in a short amount of time and helps avoiding the repetitive parts
- It is written using Python and is an open source framework
- It is based on the concept of Model-View-Controller where the data and its code (model) are independent of the user interface (view) which in turn is independent of the request logic (controller)

Overview Contd..

- It consists of an object relational mapper where the models (a python class) maps to database table giving a free dynamic database API
- Django provides an Admin interface which lets user to create, edit and delete instances of the models
- It uses a URL dispatcher which points the user to the view written for that URL
- Django provides the option of caching which allows saving of dynamic web pages
- It supports multi-language internalization of code and templates
- Used for applications like washingtonpost.com, lawrence.com, PINAX, RapidSMS

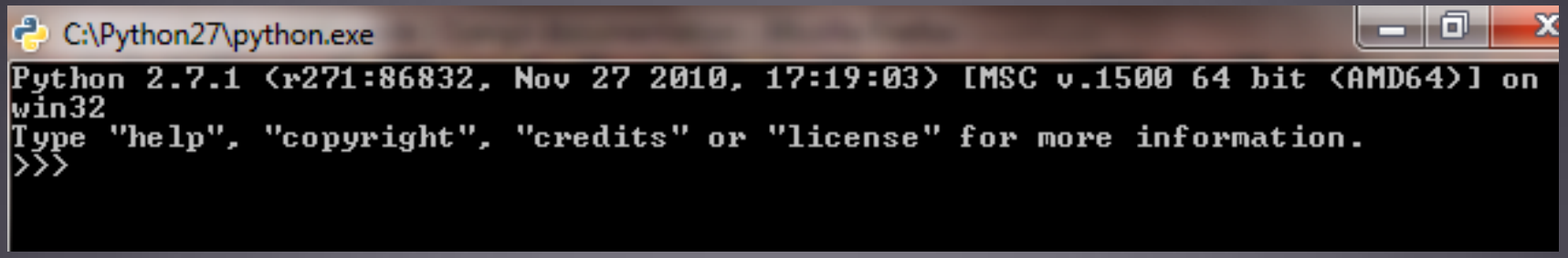
History

- Created by Adrian Holovaty and Simon Willison at the World Company in fall of 2003
- In 2005, it was released under the BSD license as an open source software
- Named after jazz guitarist Django Reinhardt
- In 2008, Django Software foundation announced the management of Django in the future

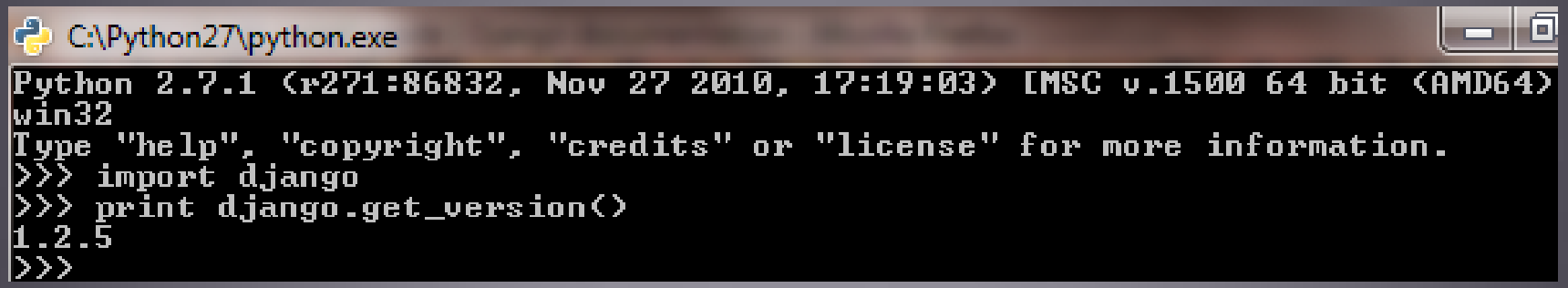


What's needed..

- It requires Python. Any version from 2.4 to 2.7 will do. To verify type python from shell



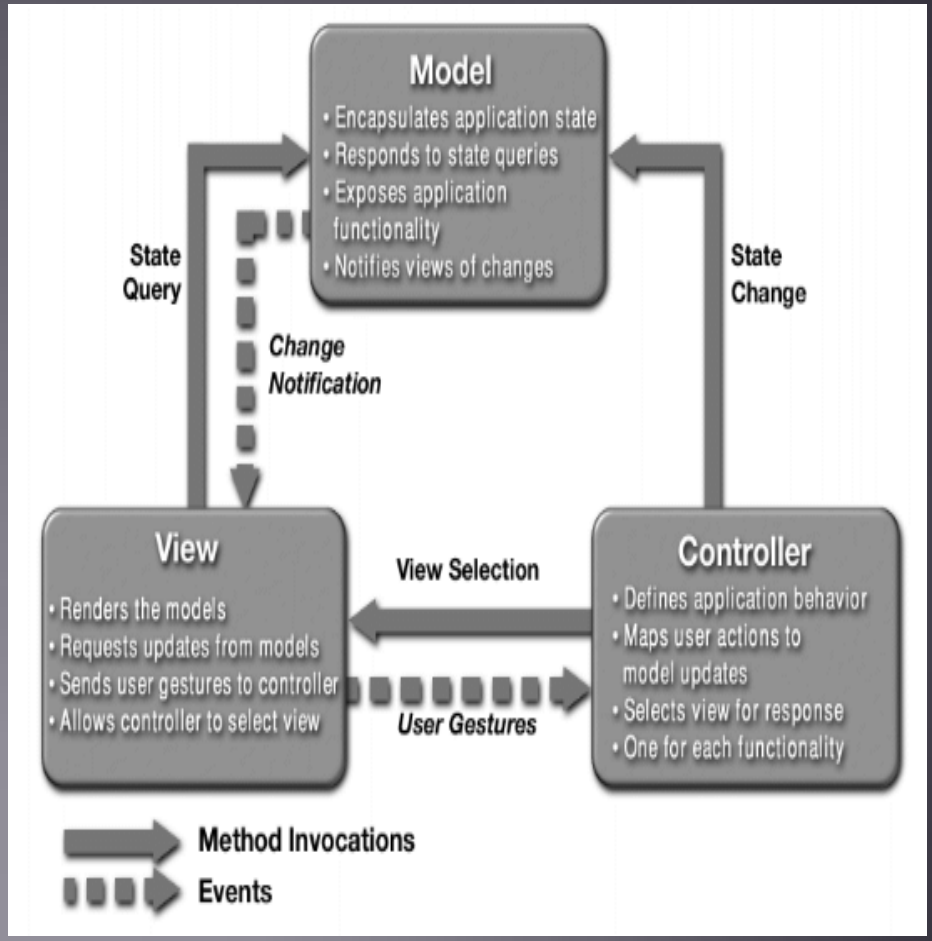
- Install Django from www.djangoproject.com



- Setup a database like PostgreSQL, MySQL, or Oracle. If Python 2.5 or later is installed, SQLite is already installed

An OO framework..

- Django uses MVC a little differently: *M-models, T-templates, V – view*
- The concept of MTV used by Django is *loosely coupled,* each is independent of the other
- It uses *inheritance* (models, templates)
- Since the modules are loosely coupled, the apps become *reusable*



Projects..

- A project is a collection of database and application settings, filesystem paths, etc. It is basically a configuration for an instance of Django

- Use the command “django-admin.py startproject *project-name*” to create a new project

- The new project created has the following four files: init__.py
  manage.py
  settings.py
  urls.py

- The init_.py file remains empty most of the times. Manage.py file is a command line utility that helps interacting with Django. Settings.py file gives the settings for the project and their default values. Urls.py gives the various urls for the project

- A project is associated with a single website

Apps..

- An app is a code that gives functionality to the website. One project can have one or many apps

- Django lets a particular app be used across various projects

- Django has many reusable apps. One of them is django.contrib:
  django.contrib.admin : automatic admin interface
  django.contrib.auth : users and groups
  django.contrib.sites: lets manage multiple sites with one Django install

- Since these apps make use of database tables, it is necessary to create a table using the command “python manage.py syncdb”

- To create an app, use “python manage.py startapp *app-name*”

- This creates a directory which includes the models.py and views.py files

Apps Contd..

- Some other reusable apps are:
  Django-messages
  Django-forum
  Django-wiki
  Django-voting

- In the setting.py file, the INSTALLED_APPS list contains these 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',
)
```

Models..

- A model is the source which describes the data
- It describes what types of data are available in the application and what fields and attributes the object has
- Each model maps to one database table
- Once you've installed your models, Django provides a highlevel API for working with them.
- Since Django believes in reuse, it wants the user to create the model not in SQL or any other database but in Python itself
- This is good because having the code in Python rather than in database gives the user better version control of the models
- Also, writing code in Python is easier than writing code in SQL

Models Contd..

- A simple code for a model may look like this. Here, we define a class for a person with first name, last name and e-mail fields

File Edit Format Run Options Windows Help

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()
```

- Each model is a Python subclass where Model is the parent class. The model class has the code to interact with the database
- Each attribute on the model corresponds to a column in the database table associated with that model
- Here, attribute name corresponds to name of column in the database and the type of field corresponds to the type of the column

Models Contd..

- Django allows models to have relationships

```
File  Edit  Format  Run  Options  Windows  Help
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()

class Article(models.Model):
    title = models.CharField(max_length=300)
    date = models.DateTimeField()
    author = models.ForeignKey(Person,related_name= "articles")
```

- After this, change the INSTALLED_APPS list in settings.py to include the above app
- Use the “python manage.py syncdb” command to create all tables, set their initial value all the new apps added to the INSTALLED_APPS list
- Use objects to access that model

Views..

- Views are a type of webpage in the application.
- It is basically a Python function that a Web request and gives a Web response
- The code can be anywhere on the Python path

```
from django.http import HttpResponse

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

- The above view is a simple function which takes a request and gives a HttpResponse
- The function hello is responsible for generating the response and the next section describes how Django finds that function

URLConf..

- Django uses a clean URL dispatcher to link URL's with their functions
- This helps providing clean URL's like /news/, /stories/, etc rather than StoryPage.aspx or script.cgi?pageid=144
- Django uses URLConf to hook a URL with a view function. The urls.py file contains all the URL's present in the application
- This is how URL's are interpreted by Django when a request comes in for , say, /hello/
- 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 /hello/.

URLConf Contd..

- 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:

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

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

- The command “python manage.py runserver” is used to test to see if the url function’s

Templates..

- A template basically is a text file. It normally generates HTML but it can generate any other type like XML, CSV

- The templates consists of three main components:
  - Variables: They are of the form {{ variable_name }}. When Django encounters variable, it evaluates that and replaces it with the result. Eg: <h1>{{ person.first_name }} {{ person.last_name }}</h1>. This is replaced by first_name attribute of person object

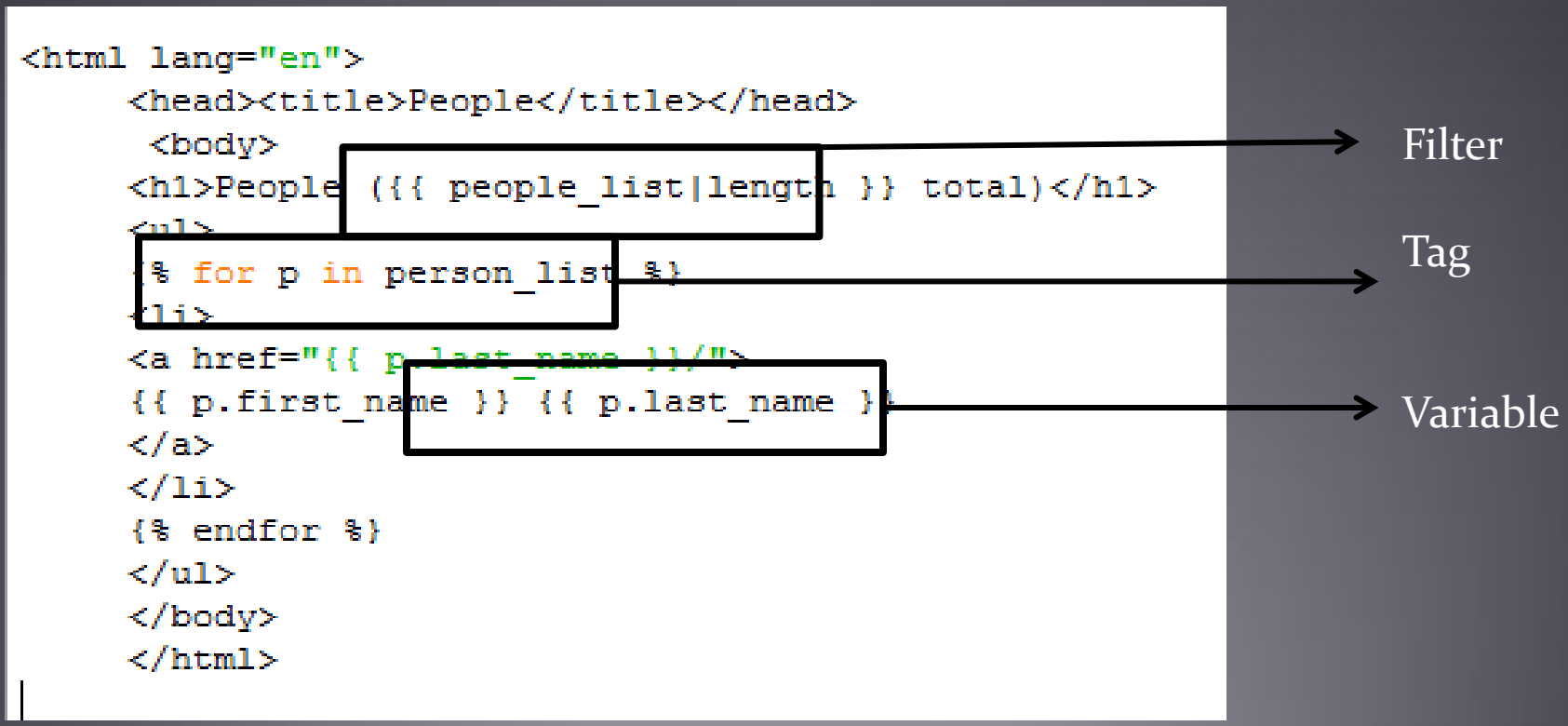
  - Tags: They are of the form {% tag_name %}. They are more complex than variables. They can produce text, loop logic. They can take in arguments.
    Eg: {% for athlete in athlete_list %}
    {{ athlete.name }}
{% endfor %}

  - Filters: They are of the form {{ variable|filter_name }}. Filters are used to modify the variable for display purposes. Eg: {{name|lower}} produces lowercase output for name variable

Templates Contd..

- A template example using variables, tags and filters is shown

```
<html lang="en">
      <head><title>People</title></head>
       <body>
      <h1>People ({{ people_list|length }} total)</h1>
      <ul>
      {% for p in person_list %}
      <li>
      <a href="{{ p.last_name }}/">
      {{ p.first_name }} {{ p.last_name }}
      </a>
      </li>
      {% endfor %}
      </ul>
      </body>
      </html>
```



Admin..

- The django admin is an optional feature provided by Django for site managers to create, update model instances.

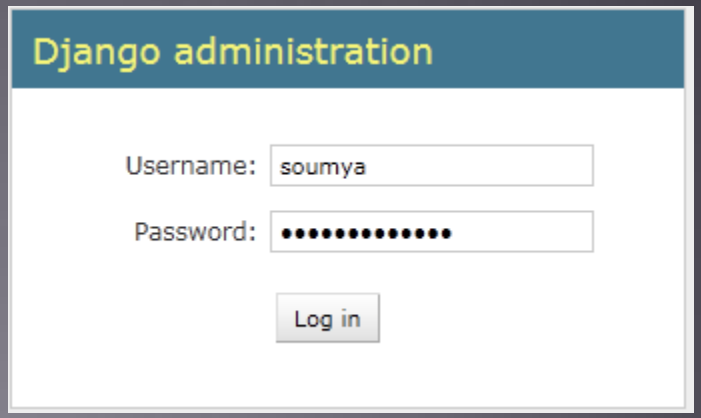
- It can work with models alone. Even if the app is not complete, you can start working with the Admin.

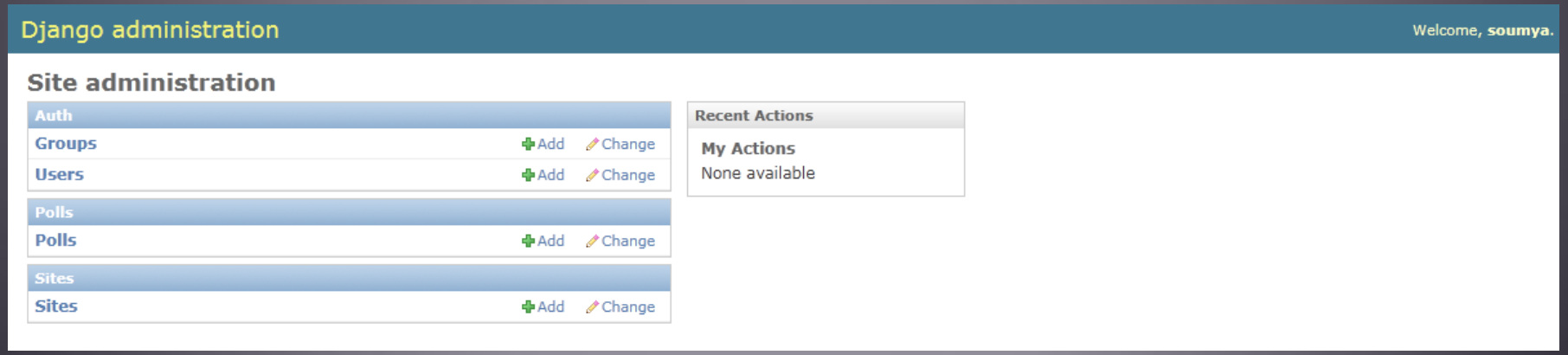
- To activate the Admin site, perform the following steps:
  - Add the contrib app- django.contrib.admin to the INSTALLED_APPS list in settings.py. Update tables using “python manage.py syncdb”
  - Edit the urls.py file to uncomment the default lines

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

urlpatterns = patterns('',
    # Example:
    # (r'^mysite/', include('mysite.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:
    #(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),
)
```

Admin Contd..

- The django admin can be run by using command “python manage.py runserver”
- My site is running at http://127.0.0.1:8000/admin/
- The login and password for the admin one which the user sets while working with Django for the first time.
- The user is free to write their own admin interface





More concepts..

- Forms: Handles HTML form display, data processing (validation) and redisplay.
- Caching: Flexible caching at any level, using memcached, database, local memory, or filesystem caching.
- Internationalization: Full support for internationalization of text.
- Middleware: a framework of hooks into Django's request/ response processing. Allow for global altering of Django's input and/or output.

Django v/s Ruby Rails

- Both are open source software
- On the template front: Django seems to have a simpler template system to help new developers

- On the Admin front: Django has an optional Admin interface which the user can use, whereas ruby on rails does not have a automatic admin page but requires 3rd party built ins
- Matter of convenience: Django runs on Python whereas Ruby on Rails runs on Ruby.

More stuff..

- Good place to start:
http://www.djangoproject.com/

- Installation help:
http://docs.djangoproject.com/en/1.3/intro/install/

- The book:
http://www.djangobook.com/

- Learn Django by example:
http://www.lightbird.net/dbe/