Belluno Linux User Group

Sviluppo di applicazioni web con

django

Belluno, 22 ottobre 2011

Parleremo di...

- Cos'è un web framework
- Cos'è Django e perché utilizzarlo
- Architettura MTV
- Modelli (database)
- Query sui dati
- Interfaccia di Admin
- URL dispatcher
- Template
- Viste ('business logic')
- Come creare un progetto

Cos'è un framework?

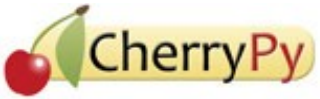
Ah, Django è un framework?

Qualcuno di voi forse si chiederà cos'è un *framework*, termine in voga e spesso usato a sproposito in ambito informatico, proprio perché molto alla moda (in inglese esiste un termine che definisce precisamente questo concetto: *buzzword*, "parola del momento"). L'oggetto di tutti i giorni che più assomiglia a un framework è un tavolo da lavoro con tutti gli attrezzi necessari: non ha uno scopo predefinito, ma fornisce i mezzi per risolvere problemi in un dato ambito. Nel nostro caso questa definizione è assolutamente azzeccata, perché Django è esattamente questo: un ottimo tavolo da lavoro, con tanto di strumenti, per costruire siti dinamici e applicazioni web.

Marco Beri - "Sviluppare applicazioni web con Django" - Apogeo 2009

- Non è un prodotto "chiavi in mano"!
- Non è un CMS (tipo Drupal, WordPress, Joomla...)!
- Serve per creare applicazioni "database driven"

Web framework Python



... e tanti altri!

Cos'è Django?

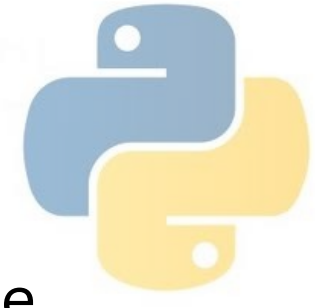
- Si pronuncia “JANG-oh” .
- Sviluppato da Holovaty, Kaplan-Moss, Willison e Miner
- Concepito inizialmente per gestire dei siti di notizie
- Storia:
  - Sviluppo iniziato nell'autunno del 2003
  - Open Source nel 2005 (licenza BSD)
  - Django 1.0 (3 settembre 2008)
  - Django 1.1 (29 luglio 2009)
  - Django 1.2 (17 maggio 2010)
  - Django 1.3 (23 marzo 2011)

NEW

- Prende il nome dal chitarrista jazz “Django Reinhardt”

Perché utilizzare Django?

"*Programming Language of the Year*" nel 2007 e 2010



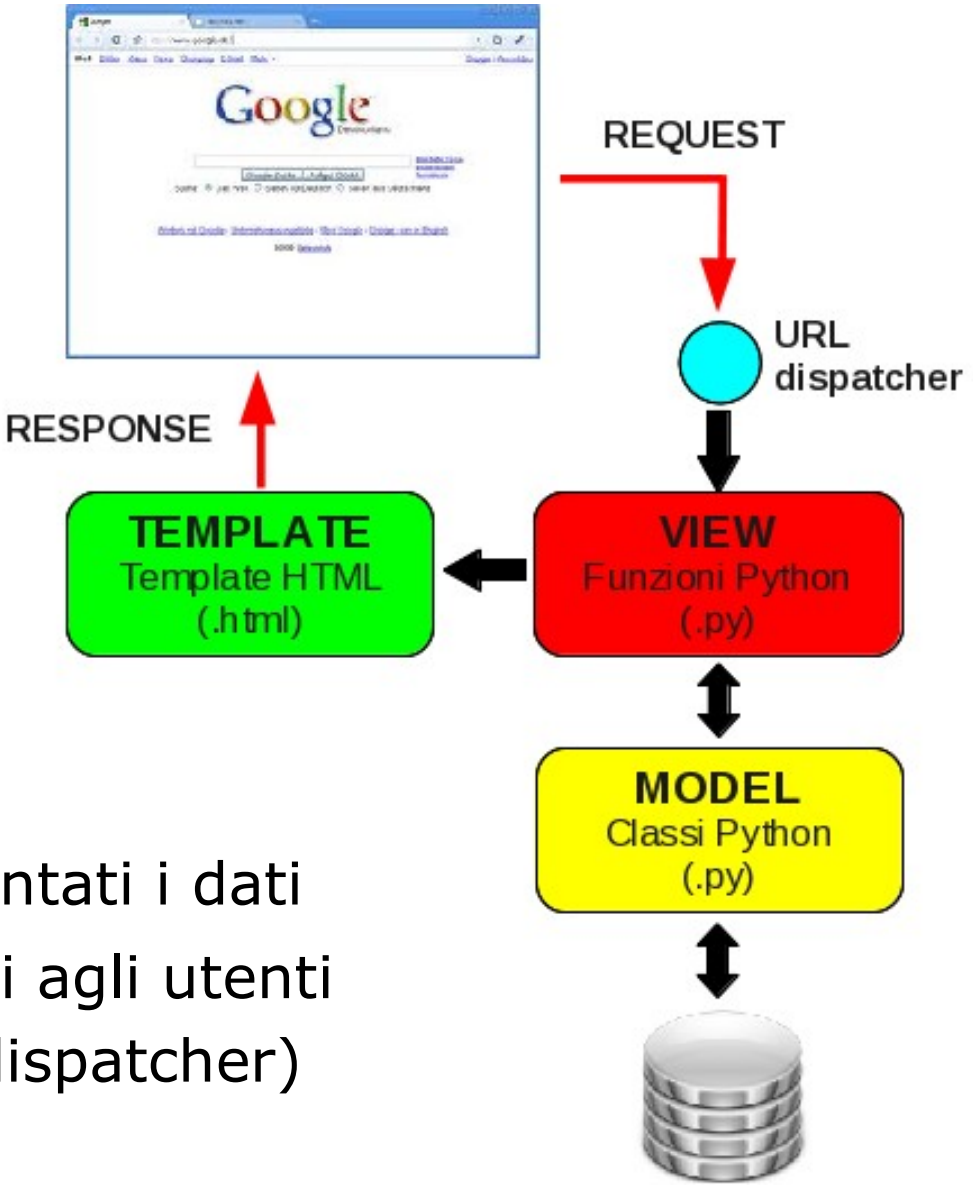
- Python powered!
- Tempi di sviluppo ridotti
- Principio DRY (Don't Repeat Yourself)
- Interfaccia di admin generata automaticamente
- Separazione della 'business logic' dalla presentazione
- Motore di templating semplice e adatto ai designer HTML
- Ottima documentazione e comunità attiva
- Localizzazione in 30+ lingue (compreso l'Italiano)
- Scalabilità e performances
- URL puliti e SEO-friendly
- Editor dedicati
- Tante applicazioni già pronte

geodjango

Architettura MTV

- Simile al modello **MVC**

"If you're hungry for acronyms, you might say that Django is a 'MTV' framework that is, 'model', 'template', and 'view'..."



- **Model**: struttura dei dati
- **Template**: COME sono presentati i dati
- **View**: QUALI dati sono inviati agli utenti
- (**Controller**: Django + URL dispatcher)

Una web application!

- Applicazione per catalogare dei libri
- Entità 'libro', 'autore' ed 'editore'
- Semplice interfaccia web
- Semplici ricerche sui dati

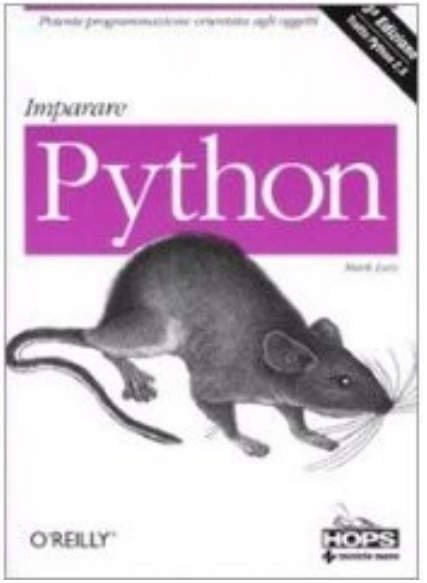
Useremo il webserver integrato!

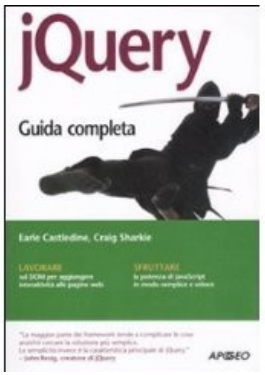
```
$ python manage.py runserver
```

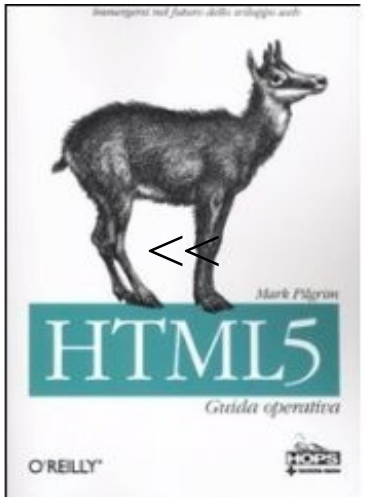
Cosa dobbiamo fare...

- Installare Django
- Creare un progetto ('mysite')
- Creare una applicazione ('books')
- Preparare il DB (no se si usa SQLite)
- Definire alcuni settaggi per il DB
- Definire i modelli dei dati
- Inserire i dati (da shell o dall'interfaccia di Admin)
- Creare la mappatura degli URL
- Definire le 'viste' (la 'business logic')
- Scrivere i 'template' (la 'presentation')
- Eseguire i necessari test
- Procedere con il deploy

Cosa serve sapere?





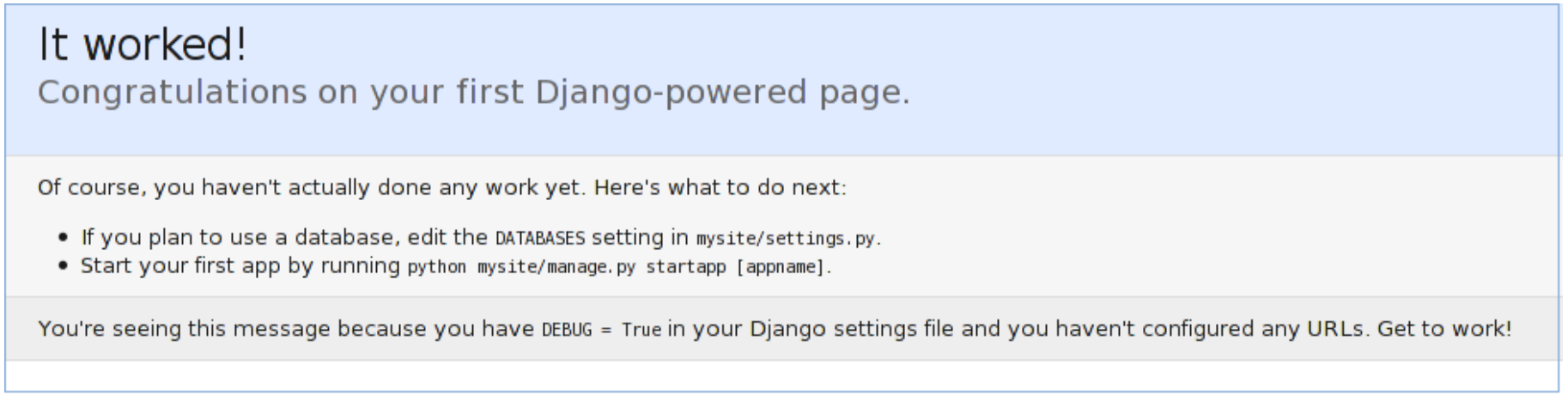


Installazione

- Unico prerequisito è Python (2.5-2.7)
- Django non funziona (ancora) con Python 3.x

1) Con il gestore di pacchetti della propria distribuzione
2) Scaricare la versione 'ufficiale' dal sito ed installarla con il comando 'python setup.py install'
3) Tramite easy_install con 'easy_install Django'

```
$ python manage.py runserver
```



Cominciamo!

Creiamo un progetto

```
$ django-admin.py startproject mysite
```

```
mysite/
|-- __init__.py
|-- manage.py
|-- settings.py
`-- urls.py
```

CONFIGURAZIONE

URL MAPPING

Creiamo una applicazione

```
$ cd mysite
$ python manage.py startapp books
```

```
books/
|-- __init__.py
|-- models.py
|-- tests.py
`-- views.py
```

MODELLI DEI DATI

BUSINESS LOGIC

Qualche configurazione

settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'mysite.db',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}

TIME_ZONE = 'Europe/Rome'
LANGUAGE_CODE = 'it-IT'

INSTALLED_APPS = (
    ...
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    ...
    # La nostra applicazione
    'mysite.books',
)
```

DATABASE

i18n

APPLICAZIONI

I modelli

- DB relazionali
- ORM (Object Relational Mapper)
- Mappatura tra classi Python e database relazionali
- Ad una classe corrisponde una tabella del DB
- Ad ogni attributo della classe corrisponde un campo del DB

```
from django.db import models

class Publisher(models.Model):
    name = models.CharField(maxlength=30)
    address = models.CharField(maxlength=50)
    city = models.CharField(maxlength=60)
    website = models.URLField()
```

ARGOMENTI

(generici e/o specifici)

TIPI DI CAMPO

(più di 20 tipi 'built-in' ed
è possibile definire tipi 'custom')

```
CREATE TABLE "books_publisher" (
    "id" serial NOT NULL PRIMARY KEY,
    "name" varchar(30) NOT NULL,
    "address" varchar(50) NOT NULL,
    "city" varchar(60) NOT NULL,
    "website" varchar(200) NOT NULL
);
```

Supporto server

- Sono supportati ufficialmente:
  - PostgreSQL
  - MySQL
  - SQLite
  - Oracle

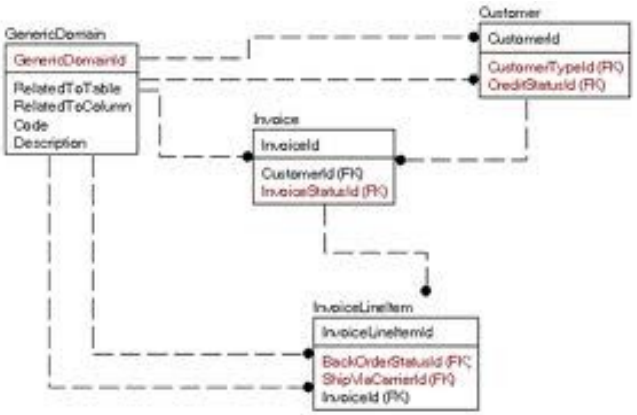
- Esistono backend non ufficiali per:
  - Microsoft SQL Server
  - IBM DB2
  - Firebird
  - NoSQL

Definire i modelli

Struttura del database

models.py

```
from django.db import models

class Publisher(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    website = models.URLField()

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

class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()
    pages = models.IntegerField()
    in_stock = models.BooleanField()
```



MOLTI A MOLTI

MOLTI A UNO

Validare i modelli

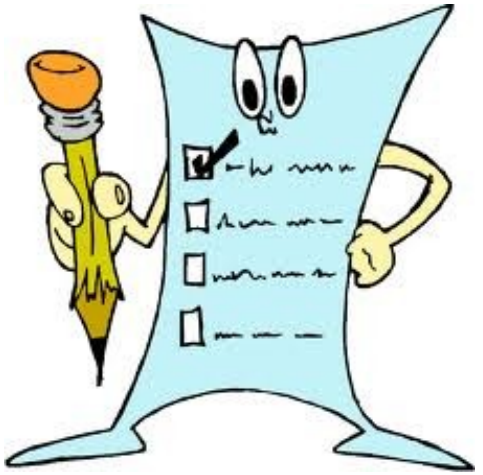
- Verifichiamo se ci sono errori

```
$ python manage.py validate
0 errors found
```

- Vediamo cosa verrà creato nel database

```
$ python manage.py sql books
BEGIN;
CREATE TABLE "books_publisher" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(30) NOT NULL,
    "address" varchar(50) NOT NULL,
    "city" varchar(60) NOT NULL,
    "website" varchar(200) NOT NULL
)
;
CREATE TABLE "books_author" (
    "id" integer NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(40) NOT NULL,
    "email" varchar(75) NOT NULL
)
;
```

continua...



Salvare il database

- Creiamo 'fisicamente' il database
- Creiamo il superuser per l'interfaccia di Admin

```
$ python manage.py syncdb
Creating tables ...
[...]
Creating table books_publisher
Creating table books_author
Creating table books_book_authors
Creating table books_book
```

TABELLE

```
You just installed Django's auth system, which means you don't have
any superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'mauro'):
E-mail address: mbarattin@libero.it
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

Modalità interattiva 1/3

- Inseriamo alcuni 'Publisher'...

```
$ python manage.py shell
Python 2.7 (r27:82500, Sep 16 2010, 18:02:00)
[GCC 4.5.1 20100907 (Red Hat 4.5.1-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)

>>> from books.models import Publisher, Author, Book
>>> import datetime

>>> apogeo = Publisher(name='Apogeo', address='Via Natale Battaglia 12',
...                    city='Milano', website='http://www.apogeonline.com')
>>> apogeo.save()

>>> oreilly = Publisher(name="O'Reilly", address='Fawcett St. 10',
...                    city='Cambridge', website='http://www.oreilly.com')
>>> oreilly.save()

>>> apress = Publisher(name='Apress', address='Telegraph Avenue 25',
...                     city='Berkeley', website='http://www.apress.com')
>>> apress.save()
```

Modalità interattiva 2/3

- Inseriamo alcuni 'Author'...

```
>>> marco = Author(first_name="Marco", last_name='Beri',
...                email='mberi@apogeo.it')
>>> marco.save()

>>> mark = Author(first_name="Mark", last_name='Luts',
...         email='mark.luts@oreilly.com')
>>> mark.save()

>>> adrian = Author(first_name="Adrian", last_name='Holovaty',
...                email='adrian@apress.com')
>>> adrian.save()

>>> jacob = Author(first_name="Jacob", last_name='Kaplan Moss',
...                email='jacob@apress.com')
>>> jacob.save()
```

Modalità interattiva 3/3

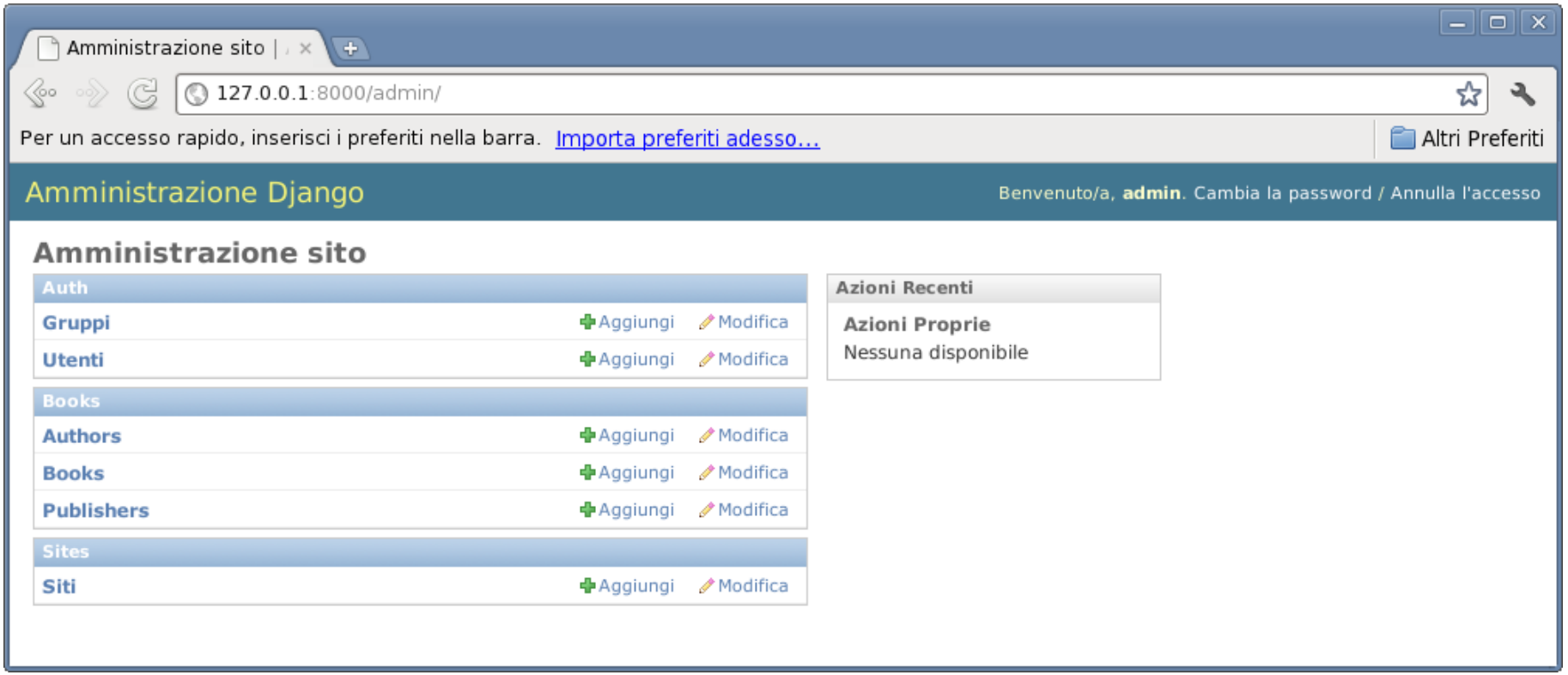
- Inseriamo alcuni 'Book'...

```
>>> book = Book(title='Sviluppare applicazioni web con Django',
...                 publisher=apogeo, publication_date=datetime.date(2009,1,16),
...                 pages=348, in_stock=True)
>>> book.save()
>>> book.authors.add(marco)

>>> book = Book(title='Programming Python',
...                 publisher=oreilly, publication_date=datetime.date(2010,12,1),
...                 pages=1632, in_stock=True)
>>> book.save()
>>> book.authors.add(mark)

>>> book = Book(title='The Definitive Guide to Django',
...                 publisher=apress, publication_date=datetime.date(2010,6,2),
...                 pages=251, in_stock=False)
>>> book.save()
>>> book.authors.add(adrian)
>>> book.authors.add(jacob)
```

L'Admin 1/4

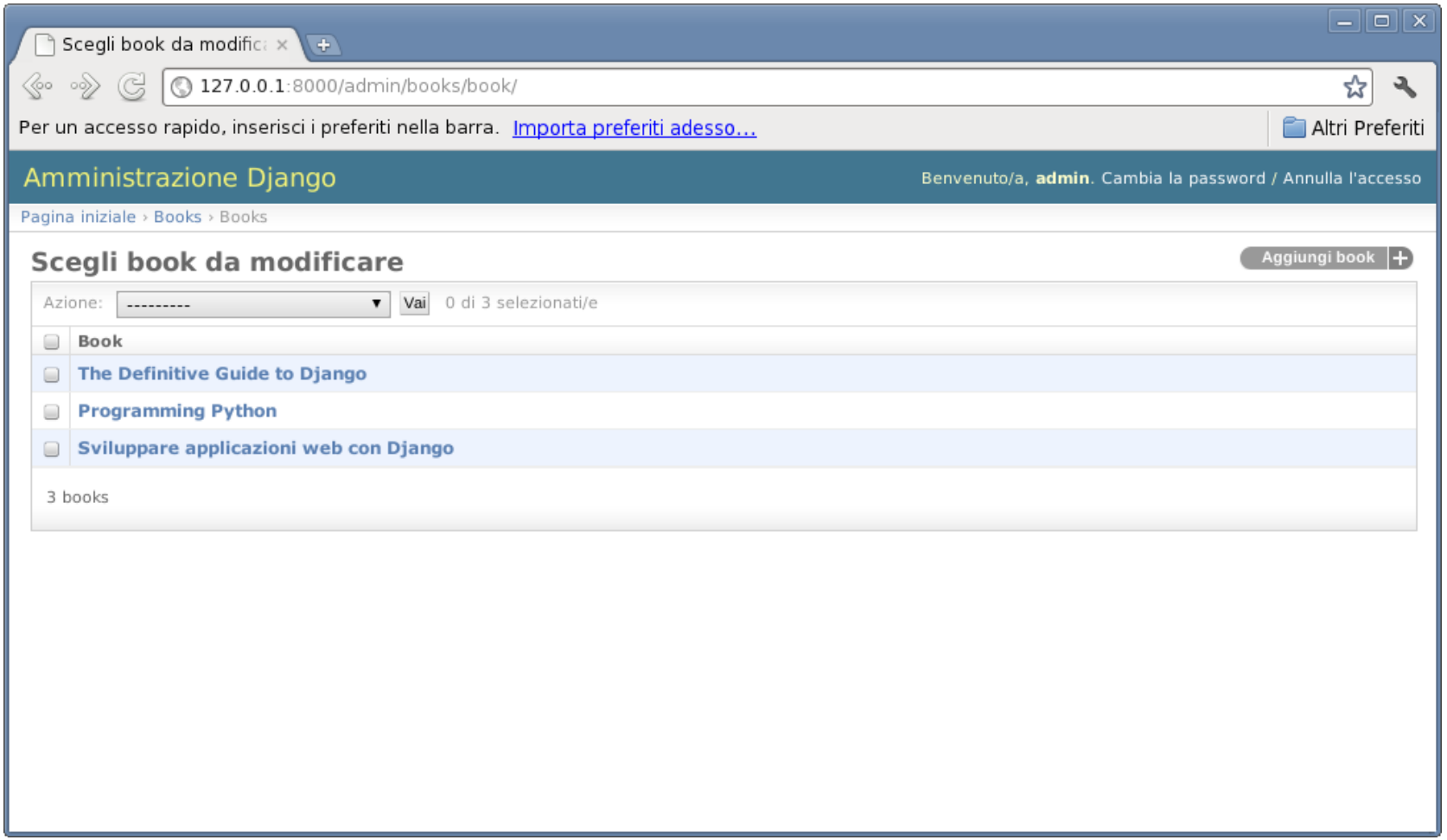
- Applicazione 'built in' di Django
- Tutto è già pronto per inserire i dati via browser!
- Basta avviare il server web di sviluppo

```
$ python manage.py runserver
```



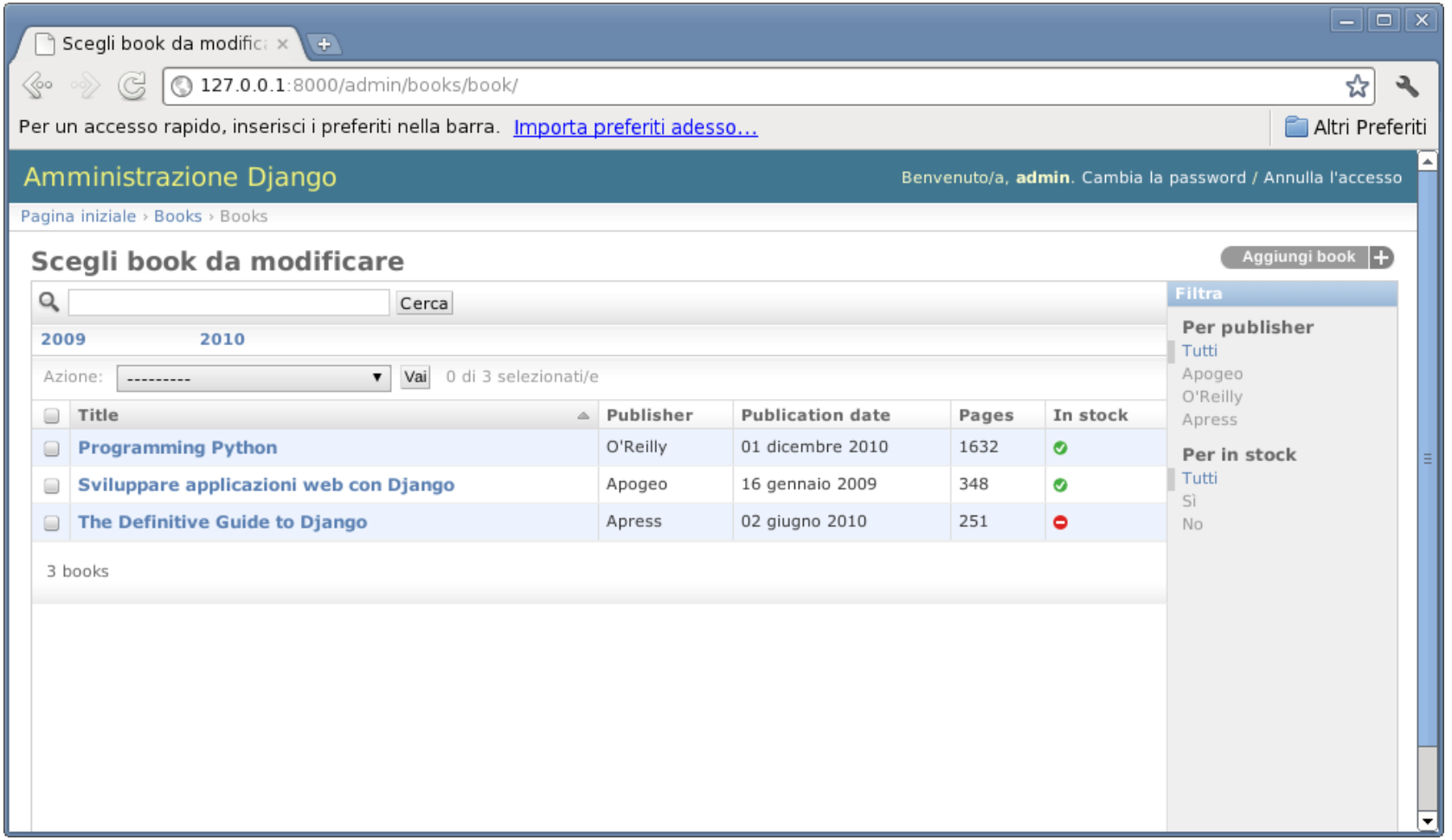
L'Admin 2/4

- Lista dei 'Books'



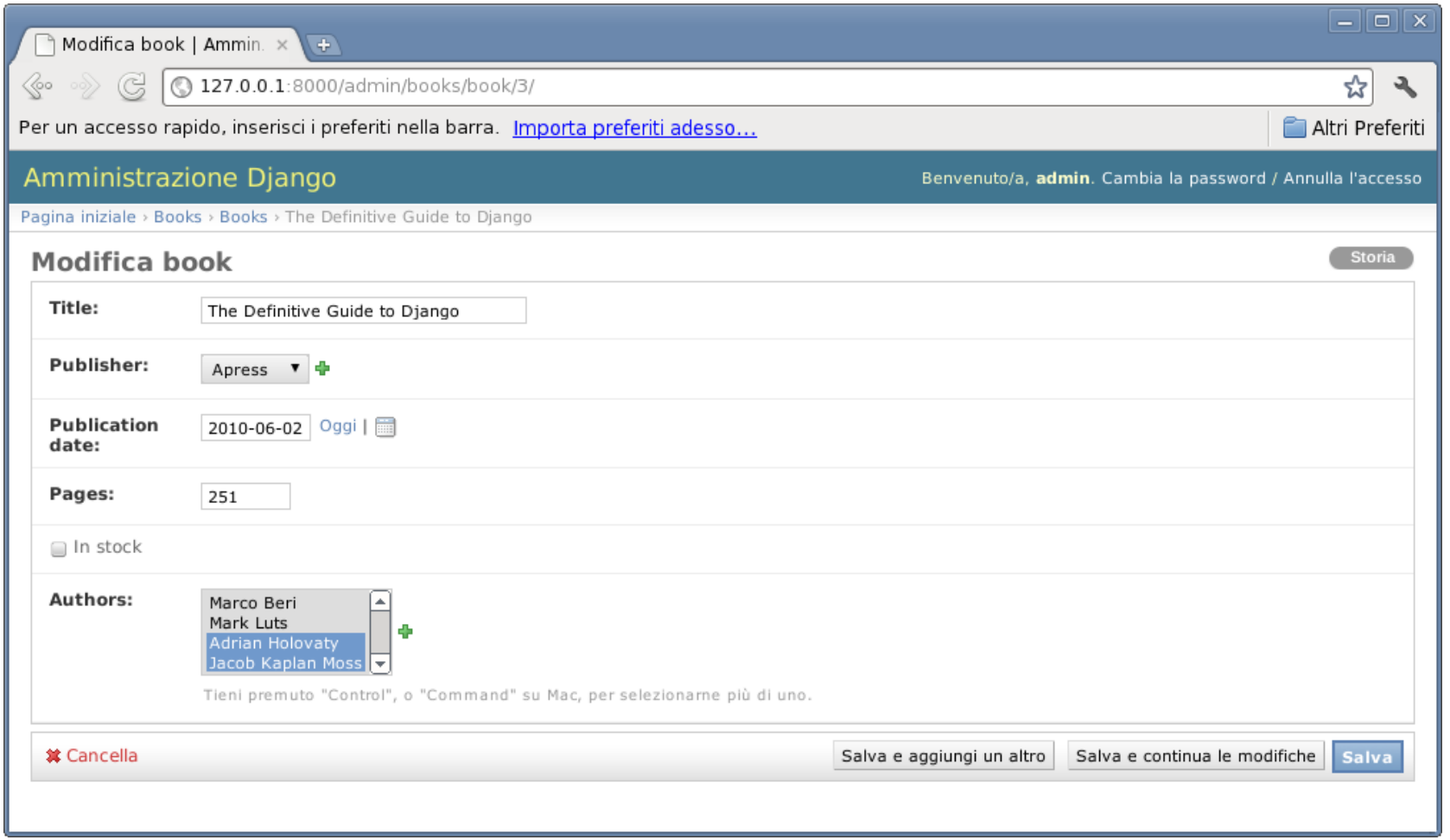
L'Admin 3/4

- Lista dei 'Books'... con un minimo di sforzo!



L'Admin 4/4

- Inserimento di un 'Books'



Selezione e filtraggio

- Si usano i metodi 'all', 'filter' e 'exclude'
- Viene restituito un QuerySet

```
>>> Book.objects.all()
[<Book: Sviluppare applicazioni web con Django>, <Book: Programming
Python>, <Book: The Definitive Guide to Django>]

>>> Book.objects.filter(pages__gte=400)
[<Book: Programming Python>]
```

Doppio 'underscore'

```
>>> Book.objects.filter(publication_date__gt=datetime.date(2010,1,1))
[<Book: Programming Python>, <Book: The Definitive Guide to Django>]

>>> Book.objects.filter(authors__first_name__istartswith="M")
[<Book: Sviluppare applicazioni web con Django>, <Book: Programming
Python>]

>>> Author.objects.filter(book__title__icontains="Django")
[<Author: Marco Beri>, <Author: Adrian Holovaty>, <Author: Jacob
Kaplan Moss>]
```

Ordinamento

- Si usano i metodi 'order_by' e 'reverse'
- Il segno '-' inverte l'ordinamento
- E' possibile specificare più di un campo

```
>>> from books.models import Publisher, Author, Book
>>> import datetime

>>> Publisher.objects.all().order_by("name")
[<Publisher: Apogeo>, <Publisher: Apress>, <Publisher: O'Reilly>]

>>> Publisher.objects.all().order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress>, <Publisher: Apogeo>]

>>> Publisher.objects.all().order_by("city", "address")
[<Publisher: Apress>, <Publisher: O'Reilly>, <Publisher: Apogeo>]
```

Cancellazione

- Si usa il metodo 'delete'
- Si applica ad un oggetto o ad un QuerySet
- Attenzione alle relazioni uno-a-molti!

```
>>> from books.models import Publisher, Author, Book

>>> p = Publisher.objects.get(name="Apress Publishing")
>>> p.delete()

>>> Publisher.objects.all().delete()

>>> Author.objects.all().delete()

>>> Book.objects.all().delete()
```

URL dispatcher

- "*Gli URL fatti bene non cambiano mai!*" - Tim Berners-Lee
- Django prevede la mappatura tra gli URL e le relative viste
- Gli URL possono avere parametri (posizionali o con nome)
- Gli URL sono "codificati" con espressioni regolari (regex)

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

urlpatterns = patterns('',

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

    # URL del nostro progetto
    url(r'^$', 'books.views.index'),
    url(r'^books/$', 'books.views.books_all'),
    url(r'^books/(\d{4})/$', 'books.views.books_year'),
    url(r'^books/(?P<year>\d{4})/$', 'books.views.books_year'),
)
```

E' la parte dopo il http://127.0.0.1:8000/

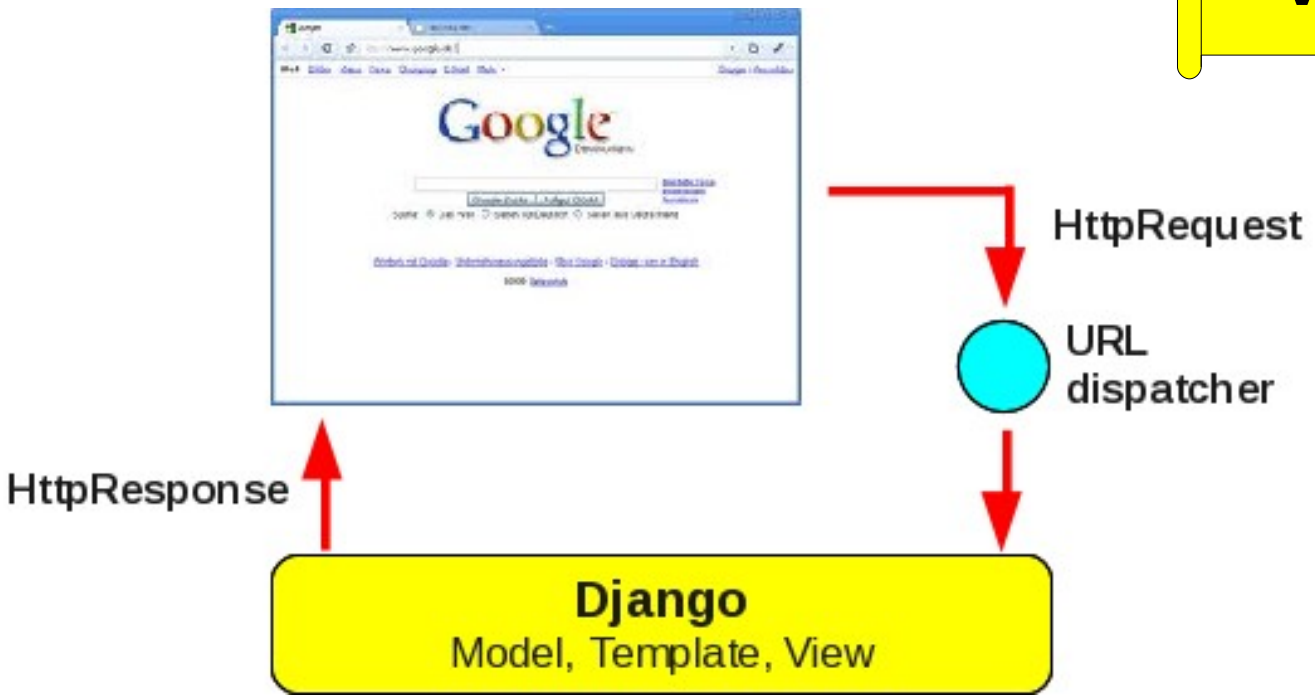
urls.py

- Viene eseguita la vista della prima espressione soddisfatta

Le viste 1/3

- La 'business logic' dell'applicazione
- Sono delle funzioni Python
- Producono i dati che vengono visualizzati dall'utente
- Ricevono (come primo parametro) un oggetto 'HttpRequest'
- Restituiscono un oggetto 'HttpResponse'

views.py



Le viste 2/3

- URL senza parametri

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import Context, RequestContext, loader
from mysite.books.models import *


def index( request ):
    return HttpResponseRedirect('/books/')

def books_all(request):

    books = Book.objects.all().order_by('title')

    t = loader.get_template("books.html")
    c = RequestContext(request, {
        'books' : books,
    })

    return HttpResponse(t.render(c))
```

OGGETTO HttpRequest

REDIREZIONE

TEMPLATE + DATI

OGGETTO HttpResponse

Le viste 3/3

- URL con parametri e qualche scorciatoia...

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import Context, RequestContext, loader
from django.shortcuts import render
from mysite.books.models import *


def books_year(request, year ):

    return  render(request, "books.html", {
              'books' : Book.objects.
                        filter(publication_date__year=int(year)).
                        order_by('title'),})
```

PARAMETRO

UNA 'SCORCIATOIA'...

I template

- Nelle viste non bisogna inserire codice HTML!
- Servono per esprimere l'aspetto (presentation)
- Dove Django cerca i template?
  - Directory “templates” nelle directory delle applicazioni
  - In alternativa valorizzare TEMPLATE_DIRS in settings.py
- Cosa contengono i template?
  - Sono file di testo (producono HTML, CSV, XML,...)
  - Istruzioni (es. istruzioni condizionali, cicli,...)
- Il linguaggio dei template è volutamente “povero”

Ereditarietà 1/2

- Ereditarietà dei template da un template “base” (base.html)
- Inclusione del contenuto di altri template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"↩
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="it" lang="it">
<head>
    <link rel="stylesheet" type="text/css"↩
       href="{{ STATIC_URL }}css/base.css" />
    <title>{% block title %}Titolo{% endblock %}</title>
</head>
<body>
    <div id="content">
    {% block content %}
    {% endblock %}
    </div>
    <div id="footer">
    {% include "footer.html" %}
    </div>
</body>
</html>
```

BLOCK

Abbondiamo con i BLOCK!!!

Ereditarietà 2/2

- I template “figli” possono ridefinire i blocchi (ma anche no...)

```
{% extends "base.html" %}
{% block title %} Book store {% endblock %}
{% block content %}
<h1>Book store</h1>
<table>
{% for book in books %}
    <tr class="{% cycle 'odd' 'even' %}">
        ...
        <td>{{ book.title }} (pag. {{ book.pages}})</td>
        <td>
            {% for author in book.authors.all %}
                <a href="mailto: {{ author.email }}">{{ author }}</a>
                {% if not forloop.last %},{% endif %}
            {% endfor %}
        </td>
        <td>{{ book.publication_date |date:"d/m/Y" }}</td>
        ...
    </tr>
{% endfor %}
</table>
{% endblock %}
```

BLOCK TAG

VARIABILI

FILTRI

Block tag

- Block tag
  - Sono compresi tra '{%' e '%}'
  - Definiscono 'logica' di un template
  - Django ha circa 30 block tag “built in”
  - E' possibile definire nuovi block tag

    Esempio: {% block *nomeblocco* %} ... {% endblock %}
    Esempio: {% comment %} ... {% endcomment %}
    Esempio: {% for *variabile* in *elenco* %} ... {% endfor %}
    Esempio: {% if *test* %} ... {% else %} ... {% endif %}
    Esempio: {% ifchanged *varibile* %} ... {% endifchanged %}
    Esempio: {% extend *nometemplate* %}
    Esempio: {% include *nometemplate* %}

- Non si va contro i dettami del paradigma MTV
- La 'logica' in un template è volutamente limitata
- La 'logica' è finalizzata solo a gestire la 'presentation'

Variabili e filtri

Variabili

- Sono comprese tra '{{' e '}}'
- Vengono (quasi sempre) passate dalle 'views'
- Non esiste la variabile: TEMPLATE_STRING_IF_INVALID

Esempio: {{ variabile }}
Esempio: {{ variabile.attributo }}

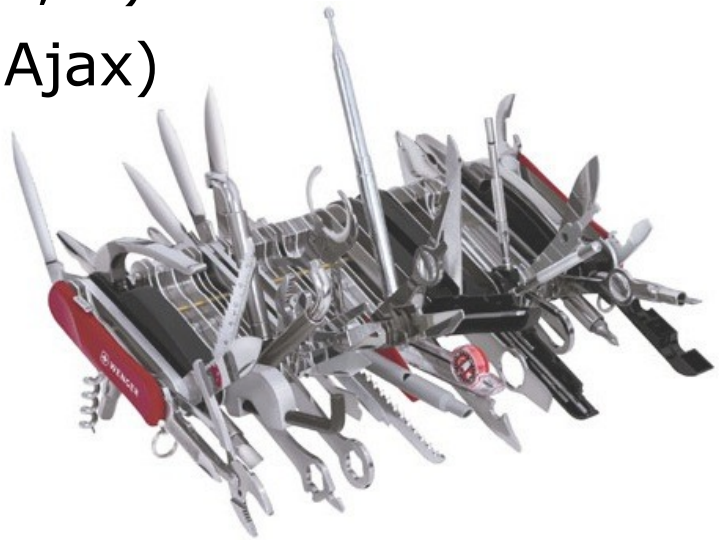
Filtri

- Modificano la visualizzazione di una variabile
- Sono separati dalla variabile da un '|' (pipe)
- Possono essere usati in cascata
- Ci sono circa 30 filtri “built in”
- E' possibile definire nuovi filtri

Esempio: {{ testo|lower }}
Esempio: {{ elenco|join:“, ” }}
Esempio: {{ testo|lower|truncatewords:10 }}

Altre 'batteries included'

- Sistema di caching
- Syndication framework (feeds RSS e Atom)
- Internazionalizzazione (i18n)
- Ottima gestione dei form (fields, widgets,...)
- Sistema di autenticazione
- Test automatici
- Generazione di file non HTML (CSV, PDF,...)
- Integrazione con le librerie Javascript (Ajax)
- Gestione delle transazioni
- Sistema dei 'middleware'
- Pagination
- Databrowse



Deployment

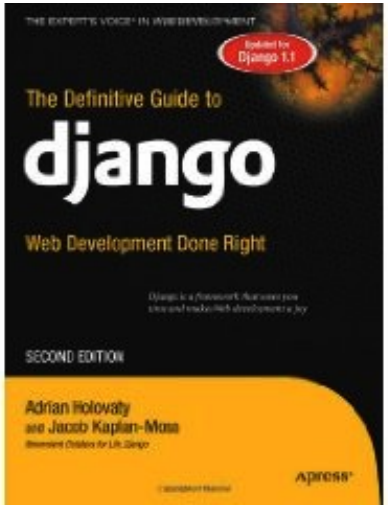
- Il server web di test non deve essere usato in produzione!
- Il metodo (fortemente) consigliato prevede l'uso di:
  - Web server **Apache**
  - Modulo di Apache **mod_wsgi**

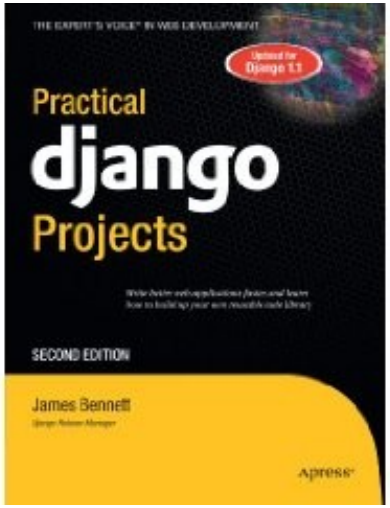
Il Web Server Gateway Interface (WSGI) è un protocollo di trasmissione che stabilisce e descrive comunicazioni ed interazioni tra server ed applicazioni web scritte nel linguaggio Python.

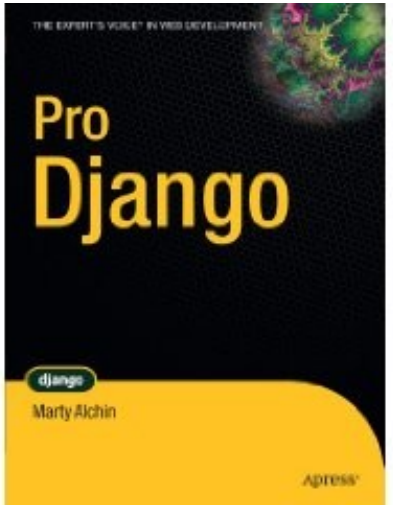
- Usare un web server che supporta FastCGI:
  - Apache (mod_fastcgi)
  - lighttpd
  - Cherokee

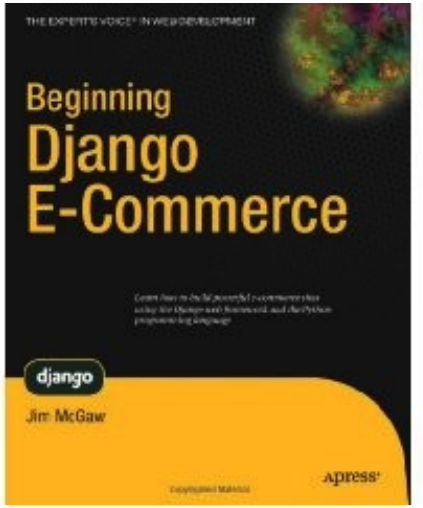
- Usare Apache con mod_python è un metodo deprecato!
- Sono supportati anche i protocolli SCGI e AJP

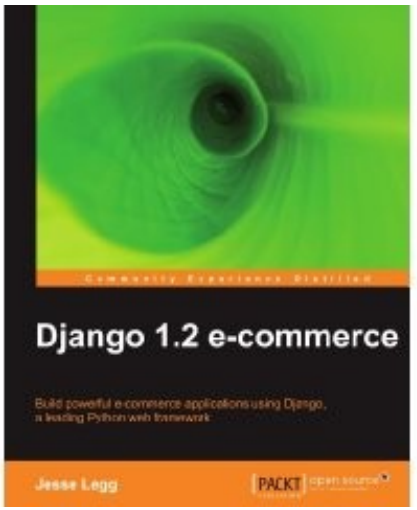
Libri su Django

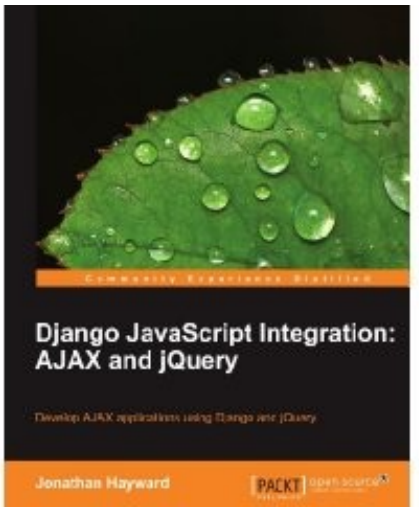


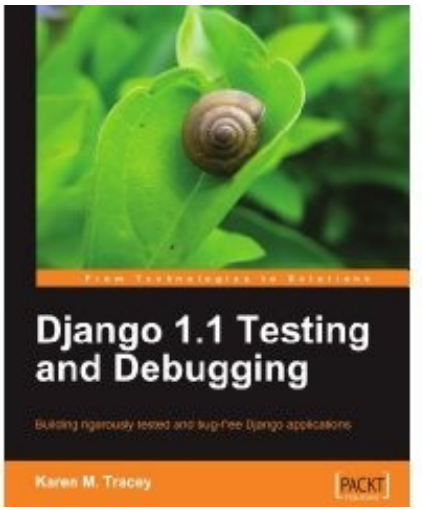












Risorse

- Il sito ufficiale (https://www.djangoproject.com)
- Pezzzetti di codice (http://djangosnippets.org)
- Applicazioni riusabili (http://djangopackages.com)
- Siti basati su Django (http://www.djangosites.org)
- Chi usa Django (http://djangopeople.net)
- Hosting per Django (http://djangofriendly.com/hosts/)
- Django Book (http://www.djangobook.com)
- Il sito italiano (http://www.djangoitalia.org)
- Mailing list (http://groups.google.com/group/django-users)
- Mailing list Italia (http://groups.google.it/group/django-it)
- IRC (irc://irc.freenode.net/django)
- Schema and data migrations (http://south.aeracode.org)
- Marco Beri - “Sviluppare applicazioni web con Django” - Apogeo 2009
- Django cheat sheet (http://www.revsys.com/django/cheatsheet/)

Grazie per l'attenzione!