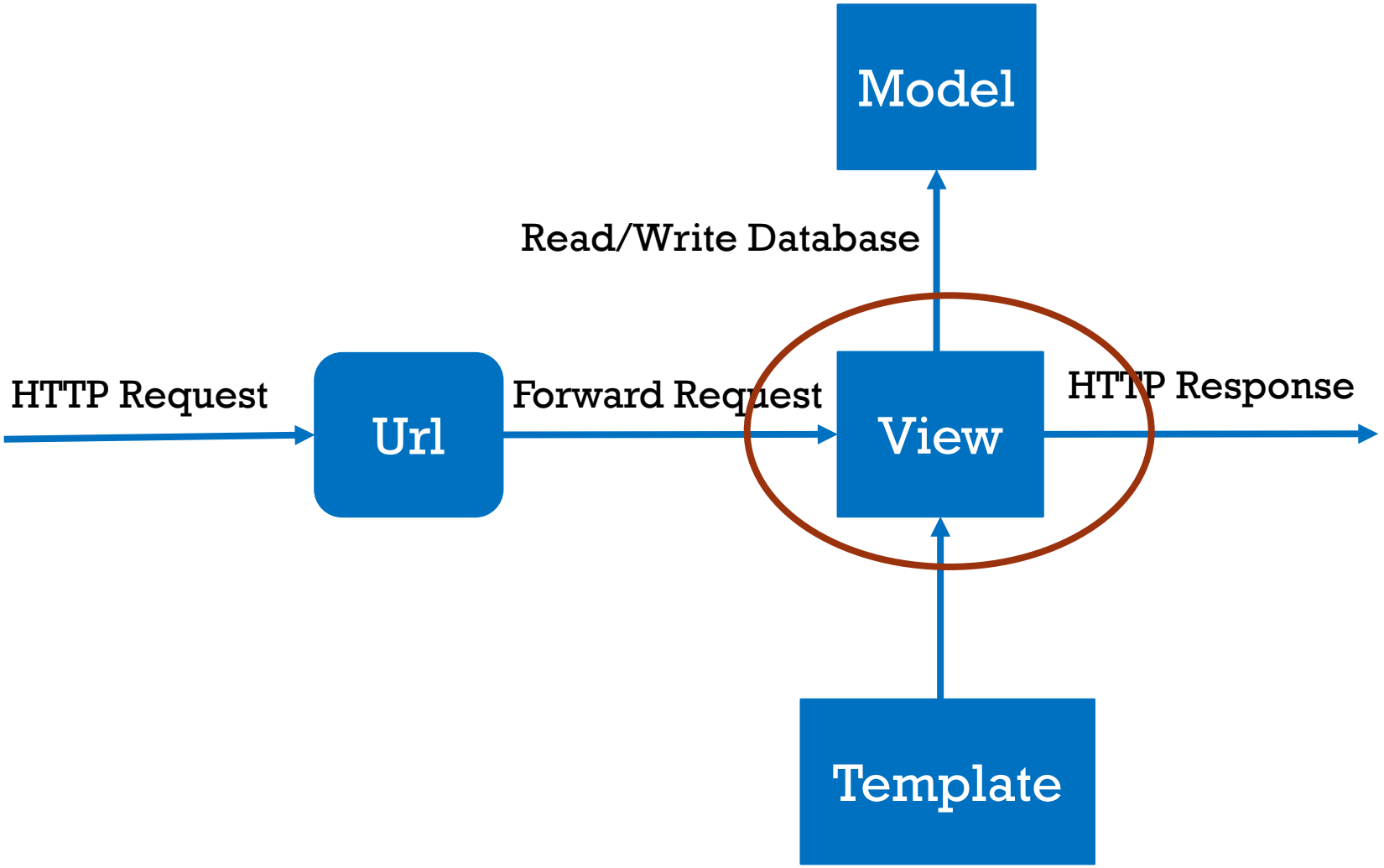
DJANGO VIEWS AND TEPLATES

江琳

VIEWS

DJANGO MVT PATTERN

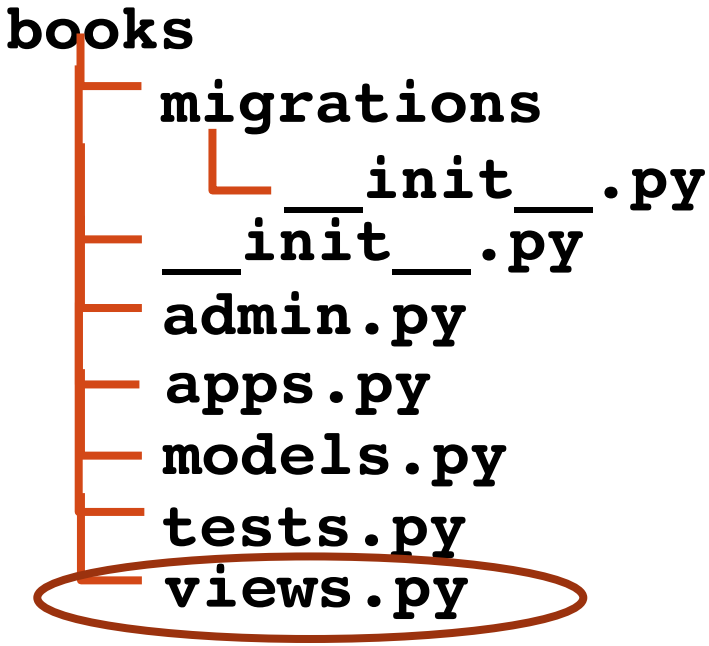


PHILOSOPHY

- **A view function, or "view" for short, is simply a Python function that takes a web request and returns a web response.**
- **This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, etc.**

VIEWS.PY

- **In Django, views have to be created in the app views.py file.**



SIMPLE VIEW

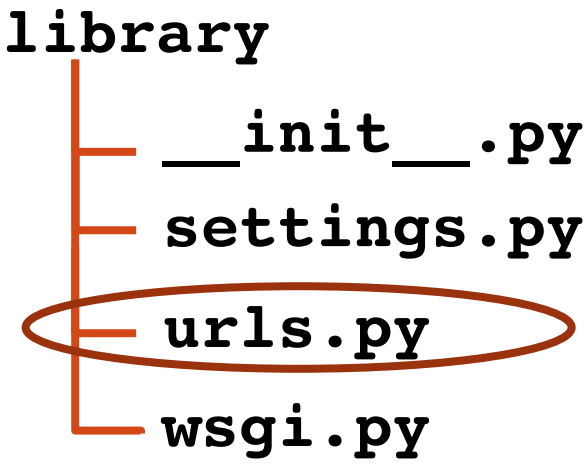
- **Coding in books/views.py**

```
from django.http import HttpResponse


def home(request):
    return HttpResponse('Welcome to Libaray!')
```

In this view, we use HttpResponse to render the HTML .
To see this view as a page we just need to map it to a URL

URL MAPPING

- **We access to view via a URL. Django has his own way for URL mapping and it's done by editing your project url.py file**



URL

```
from django.conf.urls import include, url
from books.views import home

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', home, name='home'),
]
```

When a user makes a request for a page on your web app,

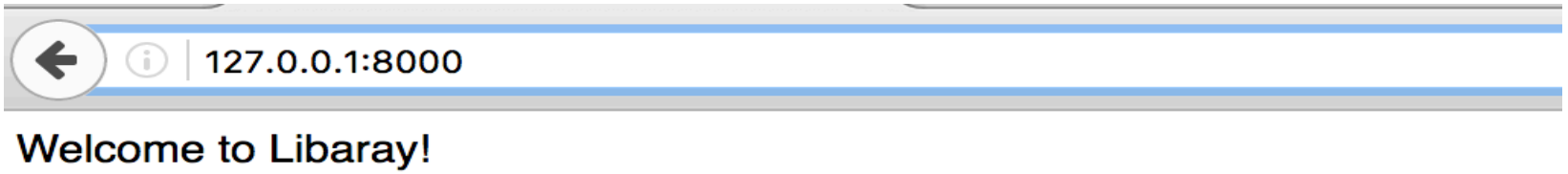
Django controller takes over to look for the corresponding

view via the url.py file.

If not fount, a 404 not fount error is returned

RUNSERVER

- **Runserver: python manage.py runserver**
- **Open link: 127.0.0.1:8000**



SIMPLE VIEW WITH PARAMETERS

- **Views can also accept parameters**

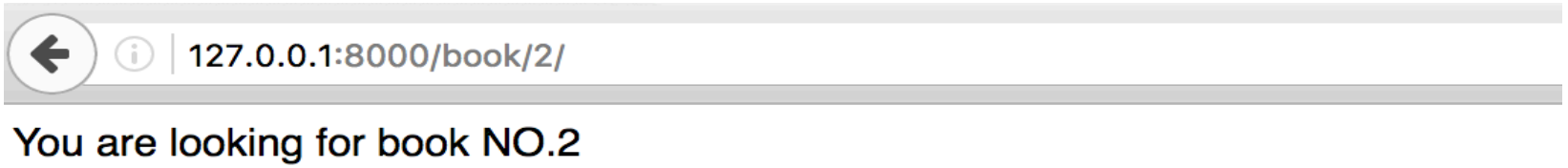
```
# in books/view.py
def BookDetail(request, id):
    return HttpResponse(
                'You are looking for book  NO.%s' % id)
```

SIMPLE VIEW WITH PARAMETERS

```
# in library/urls.py
from django.conf.urls import include, url
from django.contrib import admin

from books.views import home, BookDetail

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', home, name='home'),
    url(r'^book/(?P<id>\d+)/', BookDetail,
name='book_detail'),
]
```

SIMPLE VIEW WITH PARAMETERS

- **Open link: 127.0.0.1:8000/book/2**



SIMPLE VIEW WITH STYLE

- **Let's add some style**
  - **Modify home view in books/views.py**

```
from django.http import HttpResponse


def home(request):
    return HttpResponse('
                <h1>Welcome to Libaray!</h1>')
```

THERE IS A PROBLEM

- **You can not write all the HTML code in return**
- **Now we need Django's template system to separate the page design from Python.**

TEMPLATES

PHILOSOPHY

- **Django makes it possible to separate python and HTML, the python goes in views and HTML goes in templates. To link the two, Django relies on the render function and the Django Template language.**

PHILOSOPHY

- **TEMPLATES setting in describes how Django will load and render templates.**
- **The default settings file configures a DjangoTemplates backend whose APP_DIRS option is set to True. By convention DjangoTemplates looks for a "templates" subdirectory in each of the INSTALLED_APPS.**

SETTINTS.PY

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

TEMPLATE

- **BACKEND**: The template backend to use. The built-in template backends are:
  - 'django.template.backends.django.DjangoTemplates'
  - 'django.template.backends.jinja2.Jinja2'
- **DIRS**: Directories where the engine should look for template source files, in search order

TEMPLATE

- **APP_DIRS: Whether the engine should look for template source files inside installed applications.**
  - **Default is False**
- **OPTIONS: Extra parameters to pass to the template backend. Available parameters vary depending on the template backend.**

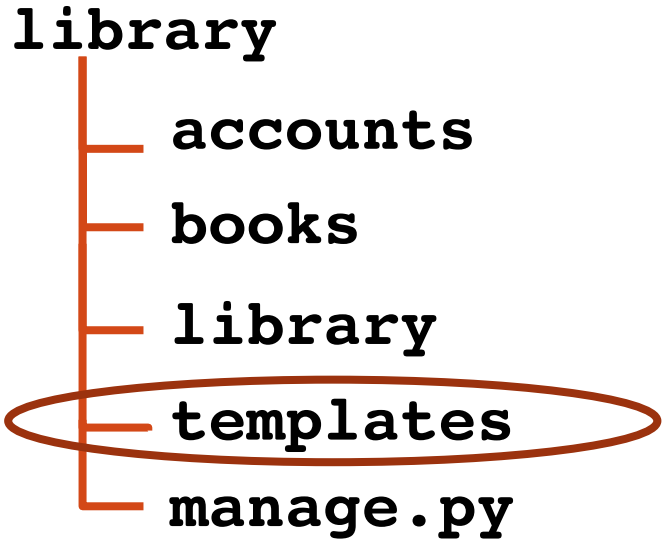
TEMPLATE

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Engine will look for template source file from this directories

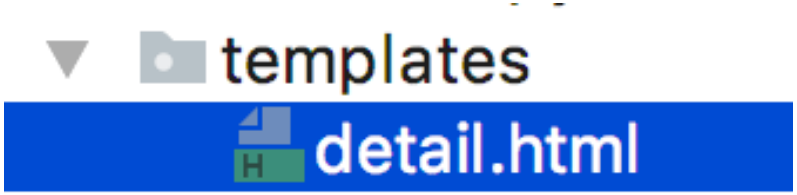
TEMPLATE

- **Let's create directory named templates in your library directory.**



FIRST TEMPLATE

- **Create a html file named detail.html inside templates folder**



FIRST TEMPLATE – VIEW.PY

- **Modify books/views.py**

```
def BookDetail(request, id):
    book = Book.objects.get(id=id)
    template = loader.get_template('detail.html')
    context = {
        'book': book
    }
    return HttpResponse(
                template.render(context, request))
```

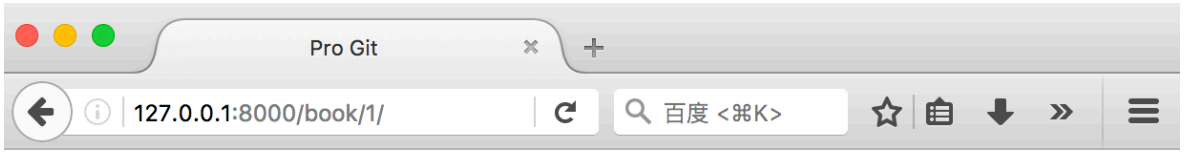
FIRST TEMPLATE

- **detail.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ book.name }}</title>
</head>
<body>
<h1>{{ book.name }}</h1>
<span>
    <b>Author:</b> {{ book.author }}
    <b>Publish by:</b>{{ book.publisher.name }}
    <b>Release at</b>:{{ book.pub_year }}
</span>
<h2>Introduction</h2>
<div>{{ book.intro }}</div>
</body>
</html>
```

FIRST TEMPLATE

- **Let's open link 127.0.0.1:8000/book/1 to see the result.**



Pro Git

Author: Scott Chacon **Publish by:**Apress **Release at**:2009

Introduction

Git is the version control system developed by Linus Torvalds for Linux kernel development. It took the open source world by storm since its inception in 2005, and is used by small development shops and giants like Google, Red Hat, and IBM, and of course many open source projects. * A book by Git experts to turn you into a Git expert * Introduces the world of distributed version control * Shows how to build a Git development workflow What you'll learn * Use Git as a programmer or a project leader. * Become a fluent Git user. * Use distributed features of Git to the full. * Acquire the ability to insert Git in the development workflow. * Migrate programming projects from other SCMs to Git. * Learn how to extend Git. This book is for all open source developers: you are bound to encounter it somewhere in the course of your working life. Proprietary software developers will appreciate Git's enormous scalability, since it is used for the Linux project, which comprises thousands of developers and testers.

DISPLAYING VARIABLES

- **A variable looks like this: {{variable}}. The template replaces the variable by the variable sent by the view in the third parameter of the render function.**

A SHORTCUT: RENDER()

- **It's a very common idiom to load a template, fill a context and return an HttpResponse object with the result of the rendered template.**
- **Django provides a shortcut.**
- **rewritten BookDetail() view :**

A SHORTCUT: RENDER()

- **Books/views.py**

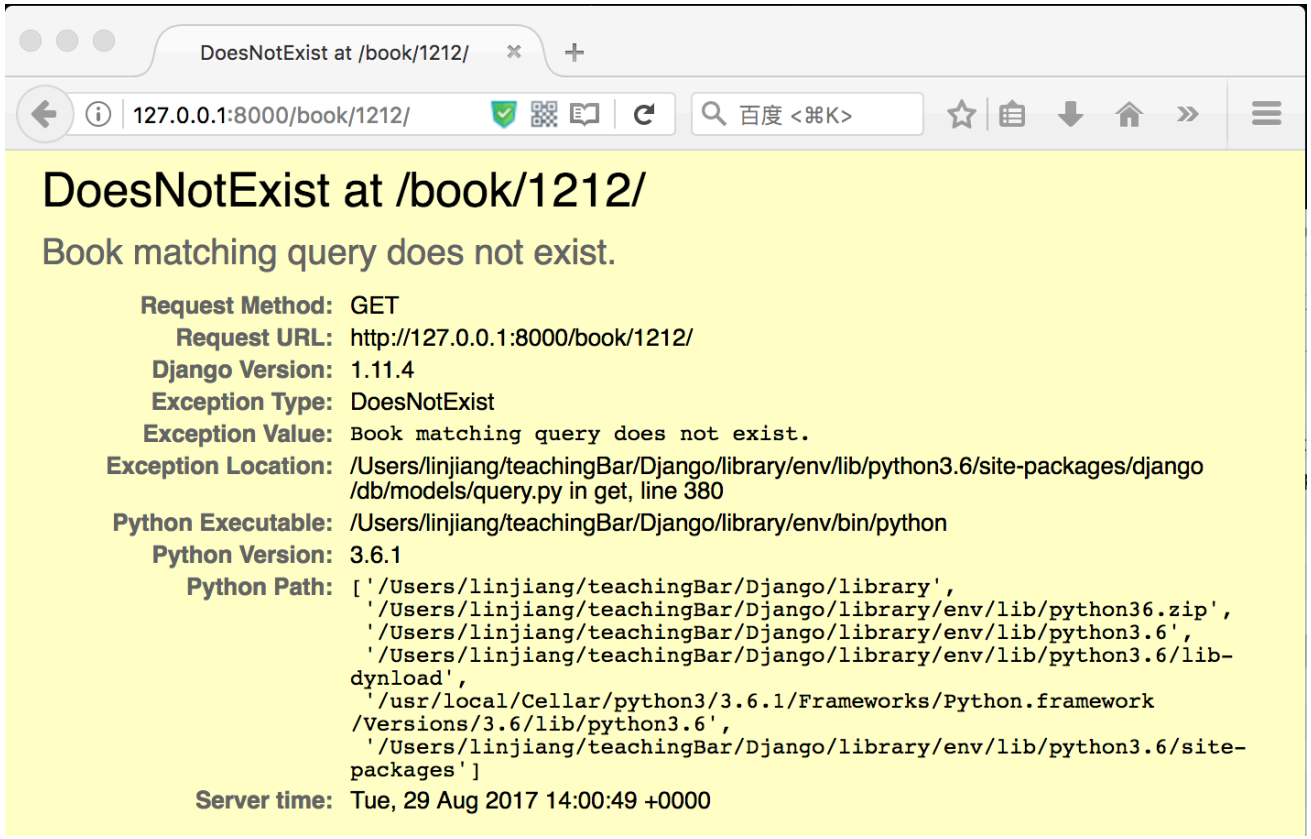
```
from django.shortcuts import render


def BookDetail(request, id):
    book = Book.objects.get(id=id)
    context = {'book': book}
    return render(request, 'detail.html', context)
```

A SHORTCUT: RENDER()

- **The render() function takes the request object as its first argument, a template name as its second argument and a dictionary as its optional third argument.**

RAISING A 404 ERROR

- **Try: 127.0.0.1:8000/book/1212**



RAISING A 404 ERROR

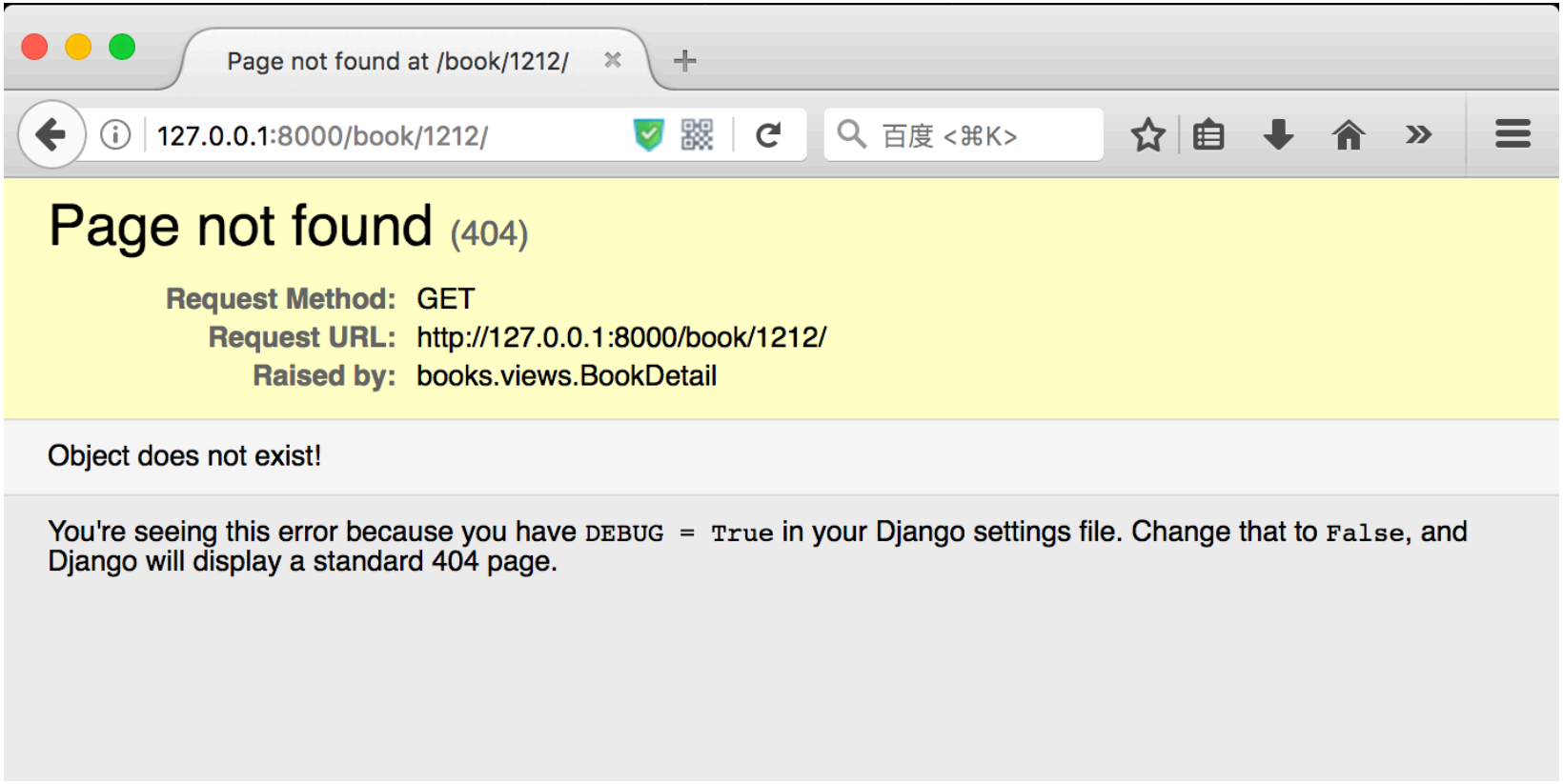
- **The view raises the Http404 exception if a question with the requested ID doesn't exist.**

RAISING A 404 ERROR

```
from django.shortcuts import render
from django.http import Http404
from .models import Book

def BookDetail(request, id):
    try:
        book = Book.objects.get(id=id)
    except Book.DoesNotExist:
        raise Http404('Object does not exist!')
    context = {'book': book}
    return render(request, 'detail.html', context)
```

- **Now open the link 127.0.0.1:8000/book/1212 again**

RAISING A 404 ERROR



A SHORTCUT: GET_OBJECT_OR_404()

- **It's a very common idiom to use get() and raise Http404 if the object doesn't exist.**
- **Django provides a shortcut get_object_or_404()**

A SHORTCUT: GET_OBJECT_OR_404()

```
from django.shortcuts import render,
get_object_or_404


def BookDetail(request, id):
    book = get_object_or_404(Book, pk=id)
    context = {'book': book}
    return render(request, 'detail.html', context)
```

LIBRARY SYSTEM

- **Assume the library system having the follow pages:**
  - **Homepage**
  - **A page list all books**
  - **A page show detail of a book**
- **Using views, templates and queryset API to write below web page.**

Questions?