Django

It's quick & easy to get up and running with Django.

Download version 1.10

Getting started with Django

Depending how new you are to Django, you can try a tutorial, or just dive into the documentation.

Want to learn more about Django? Read the overview to see whether Django is right for your project.

DJANGO OVERVIEW ›

Install Django

Before you can use Django, you'll need to install it. Our complete installation guide covers all the possibilities; this guide will get you to a simple, minimal installation that'll work while you walk through the introduction.

DJANGO INSTALLATION GUIDE ›

Write your first Django app

Installed Django already? Good. Now try this tutorial, which walks you through creating a basic poll application. It's got two parts:

1. A public site that lets people view polls and vote in them.
2. An administrative interface that lets you add, change and delete polls.

TAKE THE TUTORIAL ›

Sharpen your skills

The official Django documentation covers everything you need to know about Django (and then some).

READ THE DOCS ›

Join the community

You can help make us better. Find out about upcoming Django events, learn what's on other Django developers' minds, find and post jobs, and more.

JOIN US ›

Intro to Django

Object-relational mapper

Define your data models entirely in Python. You get a rich, dynamic database-access API for free — but you can still write SQL if needed.

READ MORE ›

```
class Band(models.Model):
    """A model of a rock band."""
    name = models.CharField(max_length=200)
    can_rock = models.BooleanField(default=True)


class Member(models.Model):
    """A model of a rock band member."""
    name = models.CharField("Member's name", max_length=200)
    instrument = models.CharField(choices=(
            ('g', "Guitar"),
            ('b', "Bass"),
            ('d', "Drums"),
        ),
        max_length=1
    )
    band = models.ForeignKey("Band")
```

URLs and views

A clean, elegant URL scheme is an important detail in a high-quality Web application. Django encourages beautiful URL design and doesn't put any cruft in URLs, like .php or .asp.

To design URLs for an application, you create a Python module called a URLconf. Like a table of contents for your app, it contains a simple mapping between URL patterns and your views.

READ MORE ›

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^bands/$', views.band_listing, name='band-list'),
    url(r'^bands/(\d+)/$', views.band_detail, name='band-detail'),
    url(r'^bands/search/$', views.band_search, name='band-search'),
]
```

```
from django.shortcuts import render

def band_listing(request):
    """A view of all bands."""
    bands = models.Band.objects.all()
    return render(request, 'bands/band_listing.html', {'bands': bands})
```

Templates

Django’s template language is designed to strike a balance between power and ease. It’s designed to feel comfortable and easy-to-learn to those used to working with HTML, like designers and front-end developers. But it is also flexible and highly extensible, allowing developers to augment the template language as needed.

READ MORE ›

```
<html>
  <head>
    <title>Band Listing</title>
  </head>
  <body>
    <h1>All Bands</h1>
    <ul>
    {% for band in bands %}
      <li>
        <h2><a href="{{ band.get_absolute_url }}">{{ band.name }}</a></h2>
        {% if band.can_rock %}<p>This band can rock!</p>{% endif %}
      </li>
    {% endfor %}
    </ul>
  </body>
</html>
```

Forms

Django provides a powerful form library that handles rendering forms as HTML, validating user-submitted data, and converting that data to native Python types. Django also provides a way to generate forms from your existing models and use those forms to create and update data.

READ MORE ›

```python
from django import forms

class BandContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

Authentication

Django comes with a full-featured and secure authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. This lets you easily build sites that let users to create accounts and safely log in/out.

READ MORE ›

```python
from django.contrib.auth.decorators import login_required
from django.shortcuts import render

@login_required
def my_protected_view(request):
    """A view that can only be accessed by logged-in users"""
    return render(request, 'protected.html', {'current_user': request.user})
```

Admin

One of the most powerful parts of Django is its automatic admin interface. It reads metadata in your models to provide a powerful and production-ready interface that content producers can immediately use to start managing content on your site. It's easy to set up and provides many hooks for customization.

READ MORE ›

```python
from django.contrib import admin
from bands.models import Band, Member

class MemberAdmin(admin.ModelAdmin):
    """Customize the look of the auto-generated admin for the Member model"""
    list_display = ('name', 'instrument')
    list_filter = ('band',)

admin.site.register(Band)  # Use the default options
admin.site.register(Member, MemberAdmin)  # Use the customized options
```

Internationalization

Django offers full support for translating text into different languages, plus locale-specific formatting of dates, times, numbers and time zones. It lets developers and template authors specify which parts of their apps should be translated or formatted for local languages and cultures, and it uses these hooks to localize Web applications for particular users according to their preferences.

READ MORE ›

```
from django.shortcuts import render
from django.utils.translation import ugettext

def homepage(request):
    """
    Shows the homepage with a welcome message that is translated in the
    user's language.
    """
    message = ugettext('Welcome to our site!')
    return render(request, 'homepage.html', {'message': message})
```

```
{% load i18n %}
<html>
  <head>
    <title>{% trans 'Homepage - Hall of Fame' %}</title>
  </head>
  <body>
    {# Translated in the view: #}
    <h1>{{ message }}</h1>
    <p>
      {% blocktrans count member_count=bands.count %}
      Here is the only band in the hall of fame:
      {% plural %}
      Here are all the {{ member_count }} bands in the hall of fame:
      {% endblocktrans %}
    </p>
    <ul>
    {% for band in bands %}
      <li>
        <h2><a href="{{ band.get_absolute_url }}">{{ band.name }}</a></h2>
        {% if band.can_rock %}<p>{% trans 'This band can rock!' %}</p>{% endif %}
      </li>
    {% endfor %}
    </ul>
  </body>
</html>
```

Security

Django provides multiple protections against:

- Clickjacking
- Cross-site scripting
- Cross Site Request Forgery (CSRF)
- SQL injection

- Remote code execution

READ MORE ›

Learn More

About Django

Getting Started with Django

Team Organization

Django Software Foundation

Code of Conduct

Diversity statement

Get Involved

Join a Group

Contribute to Django

Submit a Bug

Report a Security Issue

Follow Us

GitHub

Twitter

News RSS

Django Users Mailing List