

# RÉFÉRENCE RAPIDE DJANGO

Modèles, vues, templates, ORM, formulaires, admin, auth

## Configuration du projet

### Créer un projet et une application

```
pip install django
django-admin startproject mysite
cd mysite
python manage.py startapp blog
```

### Commandes courantes

**runserver** Démarrer le serveur de dev sur le port 8000  
**makemigrations** Générer les fichiers de migration depuis les modèles  
**migrate** Appliquer les migrations à la base de données  
**createsuperuser** Créer un superutilisateur admin  
**shell** Shell Python interactif avec Django  
**test** Exécuter la suite de tests

### Structure du projet

**manage.py** Point d'entrée CLI  
**settings.py** Configuration du projet  
**urls.py** Configuration des URL racine  
**wsgi.py / asgi.py** Points d'entrée serveur  
**apps/models.py** Modèles de base de données  
**apps/views.py** Gestionnaires de requêtes

## Modèles

### Définir un modèle

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    body = models.TextField()
    created = models.DateTimeField(auto_now_add=True)
    published = models.BooleanField(default=False)
```

### Types de champs

**CharField(max\_length=N)** Texte court (max\_length requis)  
**TextField()** Texte long (sans limite)  
**IntegerField()** Valeur entière  
**FloatField()** Nombre à virgule flottante  
**BooleanField()** Vrai / Faux  
**DateTimeField()** Date et heure  
**EmailField()** Email avec validation  
**FileField(upload\_to='')** Téléversement de fichier

### Relations

```
author = models.ForeignKey(
    User, on_delete=models.CASCADE
)
tags = models.ManyToManyField(Tag, blank=True)
profile = models.OneToOneField(User, on_delete=models.CASCADE)
```

### Meta et méthodes

```
class Meta:
    ordering = ['-created']
    verbose_name_plural = 'posts'

def __str__(self):
    return self.title
```

## Vues

### Vue basée sur une fonction

```
from django.shortcuts import render, get_object_or_404

def post_list(request):
    posts = Post.objects.filter(published=True)
    return render(request, 'blog/list.html', {'posts': posts})
```

### Vues basées sur des classes

```
from django.views.generic import ListView, DetailView

class PostListView(ListView):
    model = Post
    template_name = 'blog/list.html'
    context_object_name = 'posts'
    paginate_by = 10
```

### CBV courantes

**ListView** Afficher une liste d'objets  
**DetailView** Afficher un seul objet  
**CreateView** Formulaire pour créer un objet  
**UpdateView** Formulaire pour modifier un objet  
**DeleteView** Confirmer et supprimer un objet  
**TemplateView** Rendre un template (sans modèle)

### Réponse JSON

```
from django.http import JsonResponse

def api_posts(request):
    data = list(Post.objects.values('id', 'title'))
    return JsonResponse(data, safe=False)
```

## Templates

### Syntaxe des templates

```
{% variable %}
{% post.title|truncatewords:30 %}
{% if user.is_authenticated %}
<p>Bienvenue, {{ user.username }} !</p>
{% endif %}
```

### Boucles et conditions

```
{% for post in posts %}
<h2>{{ post.title }}</h2>
{% if forloop.last %}<hr>{% endif %}
{% empty %}
<p>Aucun article.</p>
{% endfor %}
```

### Héritage de templates

```
{% base.html #}
<html>
<body>{% block content %}{% endblock %}</body>
</html>
```

```
{% child.html #}
{% extends "base.html" %}
{% block content %}<h1>Hello</h1>{% endblock %}
```

### Filtres courants

**|date:"Y-m-d"** Formater une date  
**|default:"N/A"** Valeur de repli pour les champs vides  
**|length** Compter les éléments d'une liste  
**|truncatewords:N** Limiter à N mots  
**|safe** Marquer comme HTML sûr (pas d'échappement)  
**|slugify** Chaîne en minuscules compatible URL

## URLS

### Patterns d'URL

```
from django.urls import path, include

urlpatterns = [
    path('', views.index, name='index'),
    path('post/<int:pk>/', views.detail, name='detail'),
    path('blog/', include('blog.urls')),
]
```

### Convertisseurs de chemin

**<int:pk>** Entier (ex. 42)  
**<str:slug>** Chaîne sans barres obliques  
**<slug:slug>** Slug (lettres, chiffres, tirets)  
**<uuid:id>** Format UUID  
**<path:rest>** Chemin complet avec barres obliques

### URLS inversées

```
from django.urls import reverse
url = reverse('detail', kwargs={'pk': 1})
# Dans les templates : {% url 'detail' pk=post.pk %}
```

## Formulaires

### ModelForm

```
from django import forms

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'body', 'published']
```

### Traiter le formulaire dans la vue

```
def create_post(request):
    form = PostForm(request.POST or None)
    if form.is_valid():
        post = form.save(commit=False)
        post.author = request.user
        post.save()
        return redirect('detail', pk=post.pk)
    return render(request, 'blog/form.html', {'form': form})
```

### Formulaire dans le template

```
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Enregistrer</button>
</form>
```

### Validation

```
def clean_title(self):
    title = self.cleaned_data['title']
    if len(title) < 5:
        raise forms.ValidationError("Titre trop court.")
    return title
```

## Admin

### Enregistrer un modèle

```
from django.contrib import admin
from .models import Post
```

```
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ['title', 'author', 'created', 'published']
    list_filter = ['published', 'created']
    search_fields = ['title', 'body']
```

### Options admin

**list\_display** Colonnes dans la vue liste  
**list\_filter** Options de filtre dans la barre latérale  
**search\_fields** Champs recherchables  
**prepopulated\_fields** Auto-remplissage (ex. slug depuis le titre)  
**readonly\_fields** Non modifiables dans l'admin  
**ordering** Ordre de tri par défaut

## Requêtes ORM

### Requêtes de base

```
Post.objects.all() # tous les enregistrements
Post.objects.get(pk=1) # unique (lève une exception si absent)
Post.objects.filter(published=True) # queryset
Post.objects.exclude(draft=True) # exclure les correspondances
Post.objects.count() # nombre total
```

### Lookups de champs

**field\_exact** Correspondance exacte (par défaut)  
**field\_icontains** Contient sans distinction de casse  
**field\_gt / \_\_\_lt** Supérieur / inférieur à  
**field\_gte / \_\_\_lte** Supérieur/inférieur ou égal à  
**field\_in=[1,2,3]** Valeur dans la liste  
**field\_isnull=True** Est NULL  
**field\_startswith** Commence par une chaîne  
**field\_range=(a,b)** Entre a et b inclus

### Chaînage et agrégation

```
from django.db.models import Q, Count, Avg

Post.objects.filter(
    Q(title__icontains='django') | Q(body__icontains='django')
).order_by("-created")[:10]

Post.objects.aggregate(avg_views=Avg('views'))
```

### Créer, mettre à jour, supprimer

```
post = Post.objects.create(title='New', body='...')
post.title = 'Updated'
post.save()
Post.objects.filter(draft=True).update(published=False)
post.delete()
```

## Authentification

### Connexion / Déconnexion

```
from django.contrib.auth import authenticate, login, logout

user = authenticate(request, username='admin', password='pw')
if user is not None:
    login(request, user)
```

### Protéger les vues

```
from django.contrib.auth.decorators import login_required

@login_required
def dashboard(request):
    return render(request, 'dashboard.html')
```

### URLS d'auth

```
# urls.py
path('accounts/', include('django.contrib.auth.urls'))
# Fournit : login, logout, password_change, password_reset
```

### Auth dans les templates

```
{% if user.is_authenticated %}
<p>Bonjour, {{ user.username }}</p>
<a href="{% url 'logout' %}">Déconnexion</a>
{% else %}
<a href="{% url 'login' %}">Connexion</a>
{% endif %}
```

## Paramètres

### Paramètres clés

**DEBUG** `True` en développement, `False` en production  
**ALLOWED\_HOSTS** Liste des noms d'hôtes valides  
**SECRET\_KEY** Clé de signature cryptographique (à garder secrète)  
**DATABASES** Moteur DB, nom, hôte, identifiants  
**INSTALLED\_APPS** Liste des applications enregistrées  
**STATIC\_URL** Préfixe URL pour les fichiers statiques  
**MEDIA\_URL / MEDIA\_ROOT** Chemins des fichiers uploadés par les utilisateurs

### Config base de données

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydb',
        'USER': 'dbuser',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

### Fichiers statiques

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
# Dans les templates : {% load static %}
# <link href="{% static 'css/style.css' %}">
```