

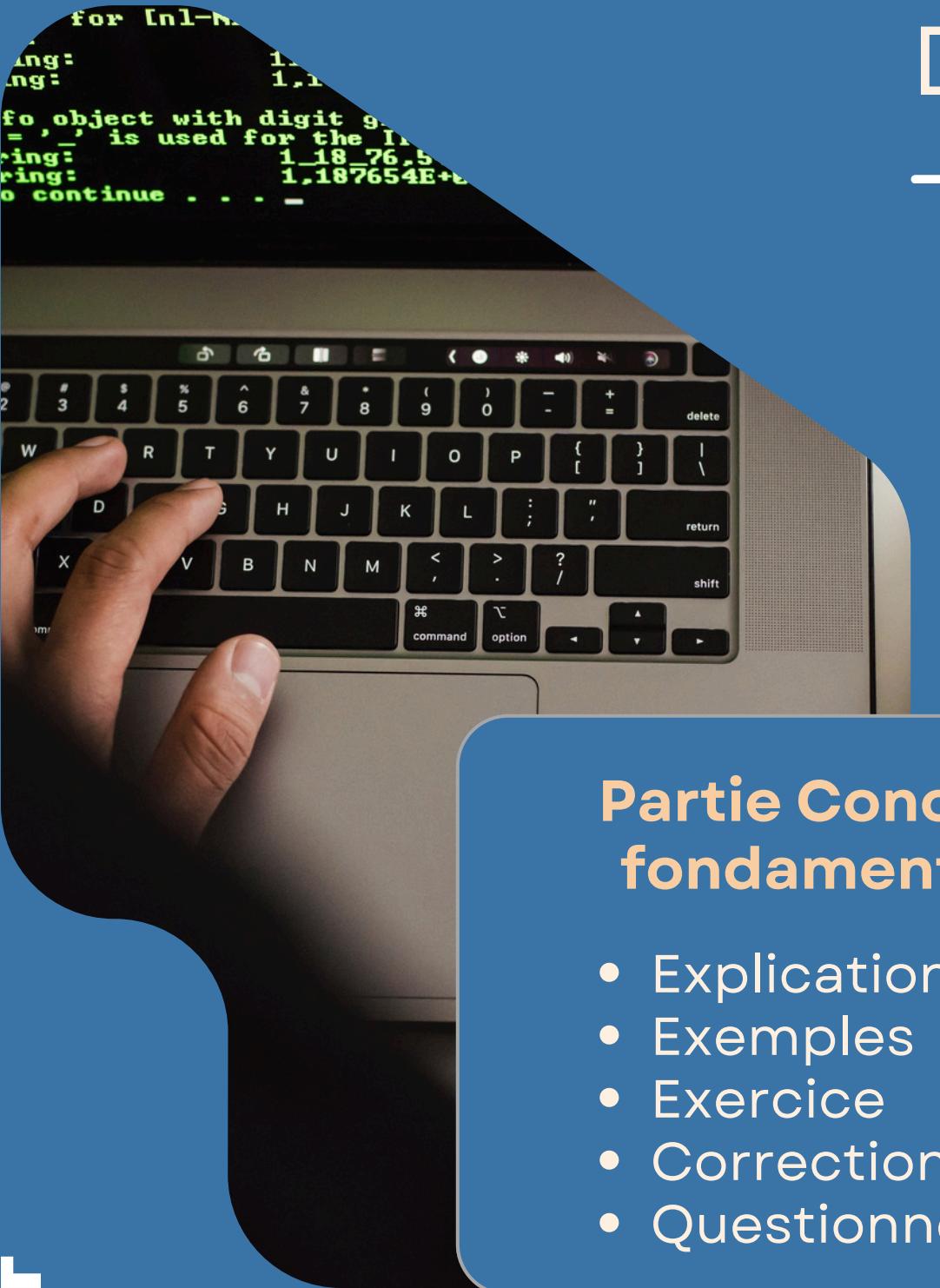
Django

Introduction au framework Python



Coffre Maureen
Logier Elsa
Ziane-Chaouche Louiza





De la théorie à la pratique



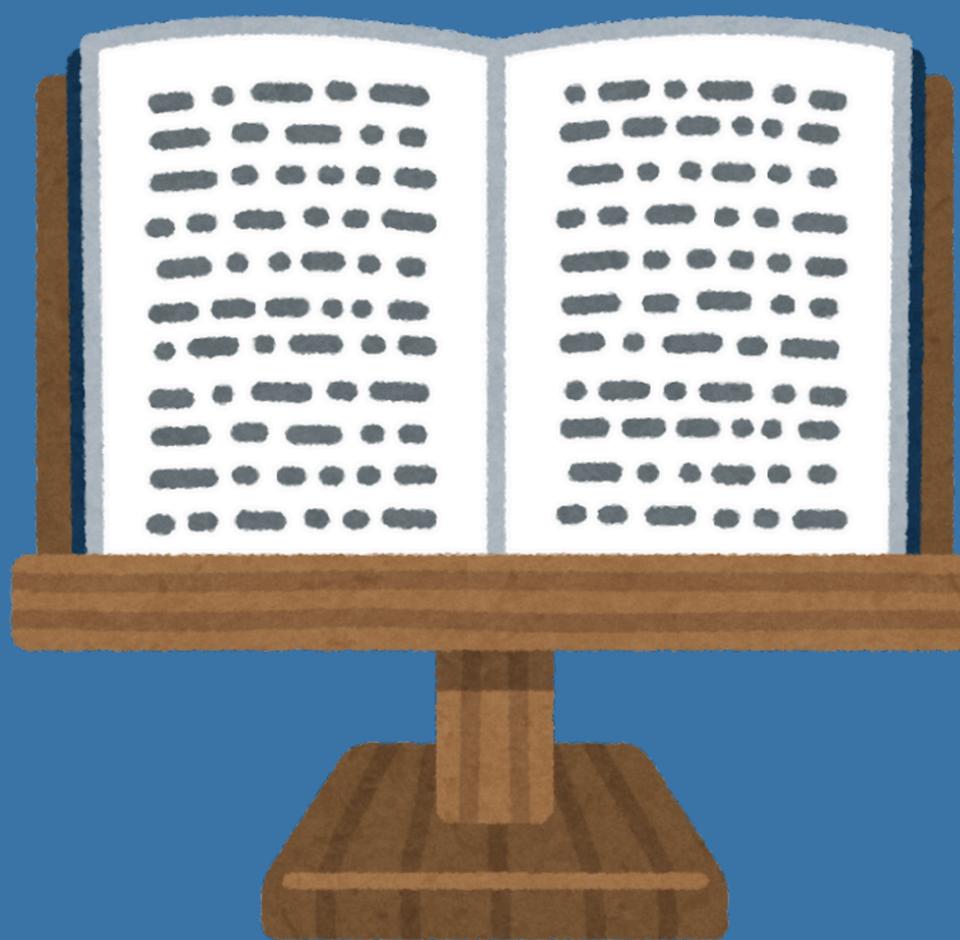
Partie Concepts fondamentaux

- Explication
- Exemples
- Exercice
- Correction
- Questionnement

- Un peu d'histoire
- Notoriété et concurrents
- Concepts fondamentaux
- Maturité et communauté
- Points forts et limites
- A retenir
- Vos feedbacks

Kahoot en fin d'exercice

Un peu d'histoire...





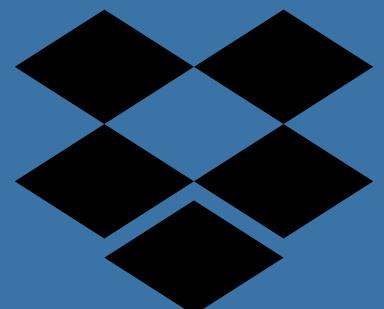
- Développé en 2003
- Pour répondre aux contraintes des journalistes
- En juillet 2005 Django a été publié sous licence BSD



Les origines du Framework !

Notoriété

Utilisation au sein de différentes entreprises



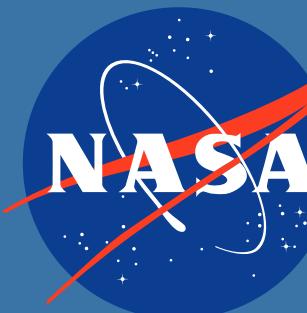
Instagram

- Traitement de données
- Interactions utilisateurs
- Augmentation efficacité du service Web



Spotify

- Backend ~ 80 %
- Bibliothèques
- Analyse de données



NASA

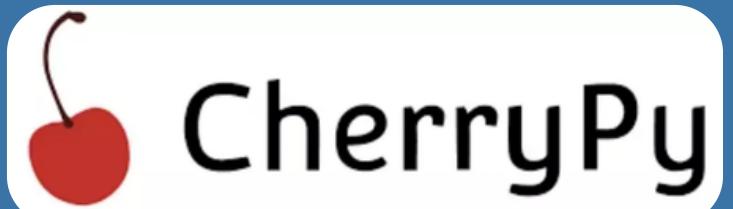
- Images haute résolution
- Contenu volumineux
- Fiabilité éléments fonctionnels WEB

Concurrents



Flask

WEB2PY



Flask

- parfait pour les petits projets ou les microservices
- minimaliste
- flexible

Tornado

- orienté applications temps réel ou asynchrones

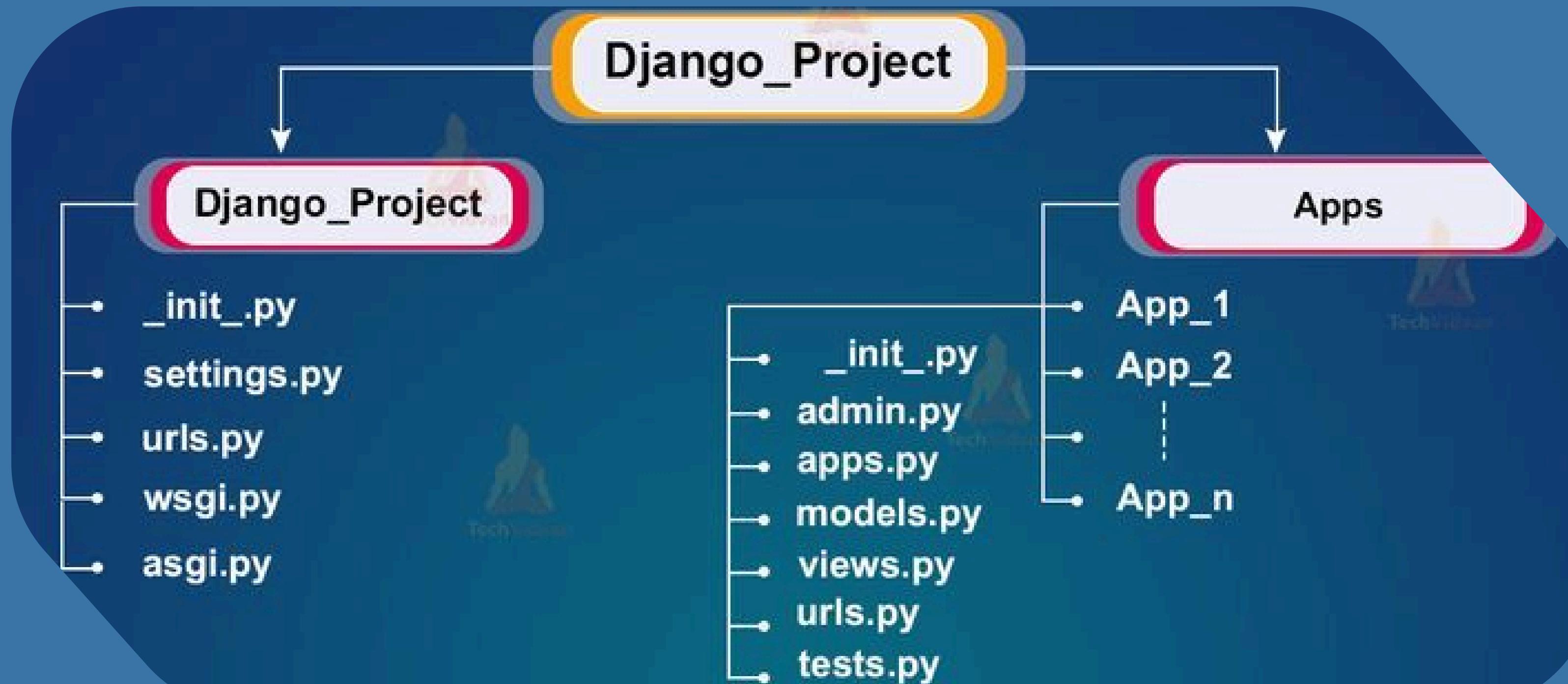
Web2py & cherryPy

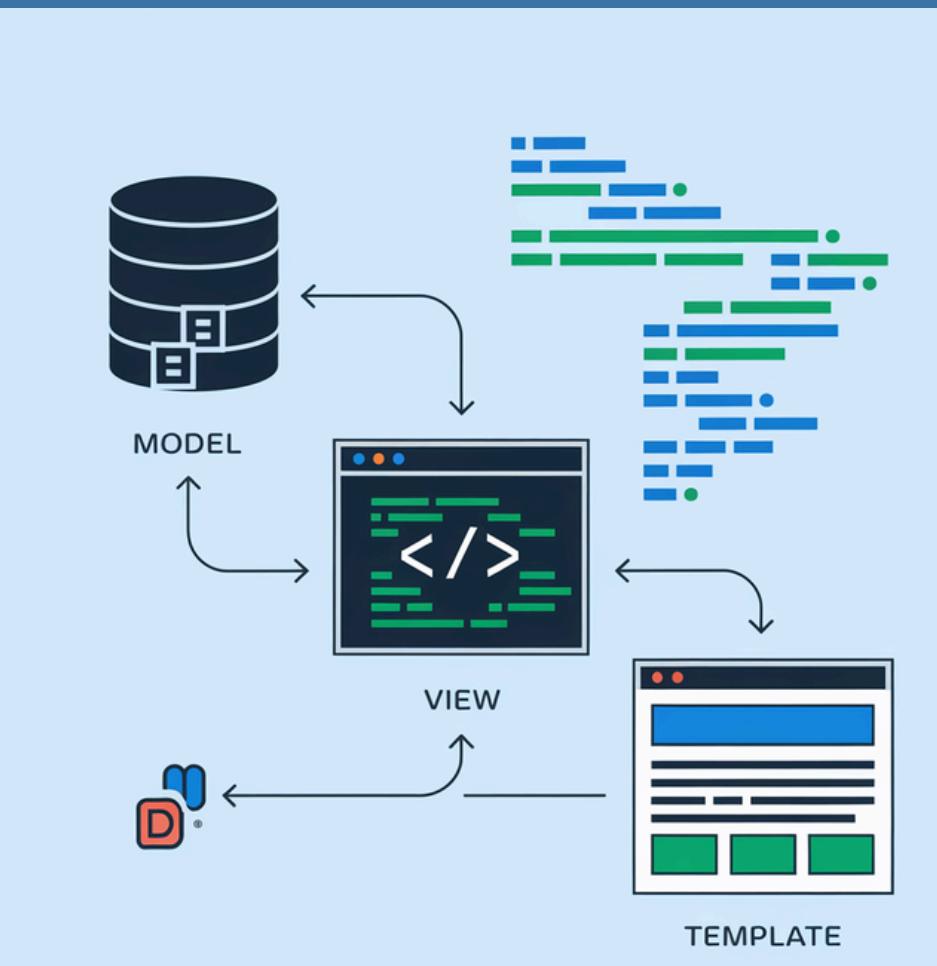
- frameworks plus anciens
- communauté moins active

Concepts Fondamentaux



Système d'applications





Le modèle MVT de Django

Django gère lui-même la partie contrôleur du MVC

Modèle (Model)
Représente la structure des données et leur logique d'accès.

Vue (View)
Reçoit la requête HTTP du client et renvoie une réponse HTTP.

Template (HTML)
Fichier HTML enrichi d'un moteur de template



Exemple

Le Modèle (Model): un modèle théorique, comme un plan d'architecte

Tickets > 📄 models.py > ...

```
1  from django.db import models  
2  
3  class Ticket(models.Model):  
4      titre = models.CharField(max_length=100)  
5      description = models.TextField()  
6      priorite = models.CharField(  
7          max_length=10,  
8          choices=[("basse", "Basse"), ("moyenne", "Moyenne"), ("haute", "Haute")],  
9          default="moyenne"  
10     )  
11     statut = models.CharField(  
12         max_length=15,  
13         choices=[("ouvert", "Ouvert"), ("en cours", "En cours"), ("fermé", "Fermé")],  
14         default="ouvert"  
15     )  
16     date_creation = models.DateTimeField(auto_now_add=True)  
17  
18     def __str__(self):  
19         return f"{self.titre} ({self.statut})"
```

→ Classe de base Django pour créer une table.

→ Définit les valeurs possibles (sélection dans un menu).

→ Remplit automatiquement la date à la création.

→ Méthode pour afficher l'objet en texte lisible dans la base.

Pour traduire le modèle Python en tables SQL dans la base., on utilise des commandes de migrations

Les migrations

■ ————— ■
Synchornisation Modèle -
BDD
■ ————— ■

■ ————— ■
Chronologie des
changements
■ ————— ■

Etape 1 : Création du fichier de migration

- Après création ou modification de modèle(s)
- Avec la commande : **python manage.py makemigrations**

Etape 2 : Application des migrations

- Exécution des requêtes SQL
- Avec la commande : **python manage.py migrate**

La Vue(View): fait le lien entre la base et l'affichage

```
Tickets > 🗃 views.py > ...
1  from django.shortcuts import render
2  from .models import Ticket
3
4  def index(request):
5      tickets = Ticket.objects.all()
6      return render(request, 'tickets/index.html', {'tickets': tickets})
7
```

The diagram shows three orange arrows originating from specific parts of the Python code in the screenshot. The first arrow points from the `render` function call to the explanatory text "Lie une vue Python à un template HTML.". The second arrow points from the `objects.all()` method call to the explanatory text "Récupère tous les tickets de la base.". The third arrow points from the `return render` statement to the explanatory text "Passe les données au template.".

Lie une vue Python à un template HTML.

Récupère tous les tickets de la base.

Passe les données au template.

La vue reçoit la requête, récupère les données, et les transmet au template.

Le Template (HTML):

```
Tickets > templates > tickets > index.html > html > body > table
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title>Backlog des tickets</title>
5  |  </head>
6  |  <body>
7  |  |  <h1> Liste des tickets</h1>
8
9  |  |  <table border="1">
10 |  |  |  <tr>
11 |  |  |  |  <th>Titre</th>
12 |  |  |  |  <th>Description</th>
13 |  |  |  |  <th>Priorité</th>
14 |  |  |  |  <th>Statut</th>
15 |  |  |  </tr>
16
17 |  |  |  &% for ticket in tickets %}
18 |  |  |  <tr>
19 |  |  |  |  <td>{{ ticket.titre }}</td>
20 |  |  |  |  <td>{{ ticket.description }}</td>
21 |  |  |  |  <td>{{ ticket.priorite }}</td>
22 |  |  |  |  <td>{{ ticket.statut }}</td>
23 |  |  |  </tr>
24 |  |  |  &% empty %
25 |  |  |  <tr><td colspan="4">Aucun ticket trouvé.</td></tr>
26 |  |  |  &% endfor %
27 |  |  </table>
28 |  |  &% load static %
29 |  |  <link rel="stylesheet" href="&% static 'tickets/style.css' %">
30 |  |  </body>
31 |  </html>
32
```

Manipuler les données avec l'ORM : Object Relational Mapping

Type d'opération	Méthode principale
Créer	<code>objects.create()</code>
Lire	<code>objects.all() / objects.get() / objects.filter()</code>
Mettre à jour	modifier les attributs → <code>save()</code>
Supprimer	<code>objects.delete()</code>



Les formulaires avec django

Plus sécurisé qu'un formulaire classique

Facilitation du développement

Vérification automatique



```
from django import forms

class MyForm(forms.Form):
    name = forms.CharField(label="Your name")
    url = forms.URLField(label="Your website", required=False)
    comment = forms.CharField()
```

Créer un formulaire manuel, indépendant d'un modèle.

Your name:

Your website:

Comment:

template.html

```
<form method="post">
    |    |    {% csrf_token %}
    |    |    {{ form.as_p }}
    |    <input type="submit" value="Valider">
</form>
```

Lier le formulaire dans la template

Your name:

Your website:

Comment:

Valider

Lier le formulaire manuel
dans dans la vue

views.py

```
class MyFormView(generic.edit.FormView):  
    template_name = "myApp/myTemplateName.html"  
    form_class = MyForm
```

On utilise un FormView

On lie la classe formulaire à la vue

```
def form_valid(self, form):  
    return super.form_valid(self, form)
```

Méthode si le formulaire est valide

```
def form_invalid(self, form):  
    return super().form_invalid(form)
```

Méthode si le formulaire est invalide

```
3 # forms.py
4 from django import forms
5 from .models import Ticket
6
7 class TicketForm(forms.ModelForm):
8     class Meta:
9         model = Ticket
10        fields = ['titre', 'description', 'sujet', 'auteur']
11
```

Le modèle auquel le formulaire est lié

Les champs à afficher dans le formulaire

Créer un formulaire lié
directement à un modèle
de la base de données.

Lier le formulaire dans la vue

views.py

```
def nouveau_ticket(request):
    if request.method == "POST":      ←
        form = TicketForm(request.POST)
        if form.is_valid():           ←
            form.save()              ←
            return redirect('liste_tickets')
    else:
        form = TicketForm()          ←
    return render(request, 'tickets/nouveau.html', {'form': form})
```

L'utilisateur a soumis le formulaire

Django vérifie automatiquement
la validité des champs

Enregistre un nouvel objet
dans la base

Formulaire vide au premier affichage

dj TESTING
✓ ✓ ✗



Les Tests dans Django

Tester, c'est prévenir les bugs avant qu'ils n'arrivent. Une application testée, c'est une application fiable.

```
class TicketModelTest(TestCase):
    def test_creation_ticket(self):
        ticket = Ticket.objects.create(
            titre="Bug de connexion",
            description="Erreur 500 sur la page login",
            auteur="Alice"
        )
```

```
        self.assertEqual(ticket.statut, "ouvert")
```

```
        self.assertEqual(str(ticket), "Bug de connexion (ouvert)")
```

Ficher # tests.py

Crée un ticket en base de données (temporaire)

Vérifie que le champ statut par défaut = "ouvert"

Vérifie que la méthode __str__ renvoie le bon texte

Exécution des tests

```
\tests> python manage.py test
```

Un allié précieux : la Django Debug Toolbar

Informations en temps réel sur le fonctionnement de l'appli

Installation : pip install django-debug-toolbar + ajout dans INSTALLED_APPS + DEBUG à true

The screenshot shows a browser window displaying the Django Debug Toolbar at localhost:8000/en/groups/. The main content area on the left provides a detailed analysis of static files, while the right sidebar contains various performance and configuration metrics.

Main Content Area (Left): Static files analysis

- Static files (162 found, 5 used)**
- Static file path**: /app/static
- Static file apps**: django.contrib.admin, rosetta, django_htmx, django_tables2, allauth.account, debug_toolbar, django_extensions
- Static files**:
 - images/favicon.png
 - css/base.css
 - js/htmx.min.js
 - images/account.png
 - js/base.js
- django.contrib.staticfiles.finders.FileSystemFinder (8 files)**

Path	Location
js/base.js	/app/static/js/base.js
js/htmx.min.js	/app/static/js/htmx.min.js
css/base.css	/app/static/css/base.css
images/account.png	/app/static/images/account.png

Toolbar (Right)

- Hide »
- Toggle Theme
- History /en/groups/ (highlighted with an orange arrow)
- Versions Django 5.1.4
- Time CPU: 115.39ms (125.81ms)
- Settings
- Headers
- Request GroupListView
- SQL 7 queries in 9.59ms
- Static files 5 files used (highlighted with an orange arrow)
- Templates groups/list.html
- Alerts
- Cache 0 calls in 0.00ms
- Signals

Détails (ici Static files)

Un outil pensé
pour les
développeurs,
pas pour les
utilisateurs
finaux



**Administration
auto-générée**

Il est auto-générée à partir des modèles déclarés dans le code Python

Personnalisation :

- Champs affichés
- Actions de masse

Système de registre : relie le modèle Python à la classe d'administration

Gestion des relations et modularité :

- ForeignKey, ManyToMany, OneToOne
- Stratégies de modularité
- Modularité applicative



Exemple

```
mysite > ⚙ settings.py > ...
37
38 INSTALLED_APPS = [
39     "polls.apps.PollsConfig",
40     'django.contrib.admin', _____ → Interface admin
41     'django.contrib.auth', _____ → Utilisateurs et groupes
42     'django.contrib.contenttypes',
43     'django.contrib.sessions',
44     'django.contrib.messages',
45     'django.contrib.staticfiles',
46     'debug_toolbar',
47 ]
48
```

Interface admin minimale 

Concepts fondamentaux



Gestion des permissions

Polls Administration

Home > Authentication and Authorization > Groups > Add group

Add group

Name:

Permissions:

Available permissions

Choose permissions by selecting them and then select the "Choose" arrow button.

Filter

Administration | log entry | Can add log entry
Administration | log entry | Can change log entry
Administration | log entry | Can delete log entry
Administration | log entry | Can view log entry
Authentication and Authorization | group | Can add group
Authentication and Authorization | group | Can change group
Authentication and Authorization | group | Can delete group
Authentication and Authorization | group | Can view group
Authentication and Authorization | permission | Can add permission
Authentication and Authorization | permission | Can change permission

Choose all permissions Hold down "Control", or "Command" on a Mac, to select more than one.

Remove all permissions

SAVE Save and add another Save and continue editing

Création d'utilisateur

Polls Administration

Home > Authentication and Authorization > Users > Add user

Add user

After you've created a user, you'll be able to edit more user options.

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#) [Change](#)

Users [+ Add](#) [Change](#)

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password-based authentication: Enabled Disabled
Whether the user will be able to authenticate using a password or not. If disabled, they may still be able to authenticate using other backends, such as Single Sign-On or LDAP.

Password: Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

SAVE Save and add another Save and continue editing

```
polls > admin.py > ...
1  from django.contrib import admin
2
3  from .models import Question, Choice ← Modèles métiers
4
5  class ChoiceInline(admin.TabularInline):
6      model = Choice
7      extra = 3 ← Edition des Choice liés à une
8
9
10 class QuestionAdmin(admin.ModelAdmin):
11     fieldsets = [ ← Structure du formulaire
12         (None, {"fields": ["question_text"]}),
13         ("Date information", {"fields": ["pub_date"], "classes": ["collapse"]}), ← d'édition
14     ]
15     inlines = [ChoiceInline]
16     list_display = ["question_text", "pub_date", "was_published_recently"] ← Colonnes visibles dans la liste
17     list_filter = ["pub_date"] ← Filtrage
18     search_fields = ["question_text"] ← Barre de recherche
19
20 admin.site.register(Question, QuestionAdmin) ← Enregistrement du modèle
21
```

Polls Administration

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	Change
Users	+ Add	Change

POLLS

Questions	+ Add	Change
-----------	-------	--------

Recent actions

My actions

- + test User
- What's up ? Question

Polls Administration

WELCOME, ADMIN. [VIEW SITE / CHANGE PASSWORD / LOG OUT](#)

Home > Polls > Questions

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add
Users	+ Add

POLLS

Questions	+ Add
-----------	-------

Select question to change

Action: ----- Go 0 of 2 selected

<input type="checkbox"/> QUESTION TEXT	DATE PUBLISHED	PUBLISHED RECENTLY?
<input type="checkbox"/> How are you ?	Oct. 27, 2025, 8:47 p.m.	<input checked="" type="checkbox"/>
<input type="checkbox"/> What's up ?	Sept. 28, 2025, 1:35 p.m.	<input type="checkbox"/>

ADD QUESTION +

FILTER

Show counts

By date published

Any date

Today

Past 7 days

This month

This year

2 questions

«

Add question

Question text:

Date information

Date published: Date: Today |

Time: Now |

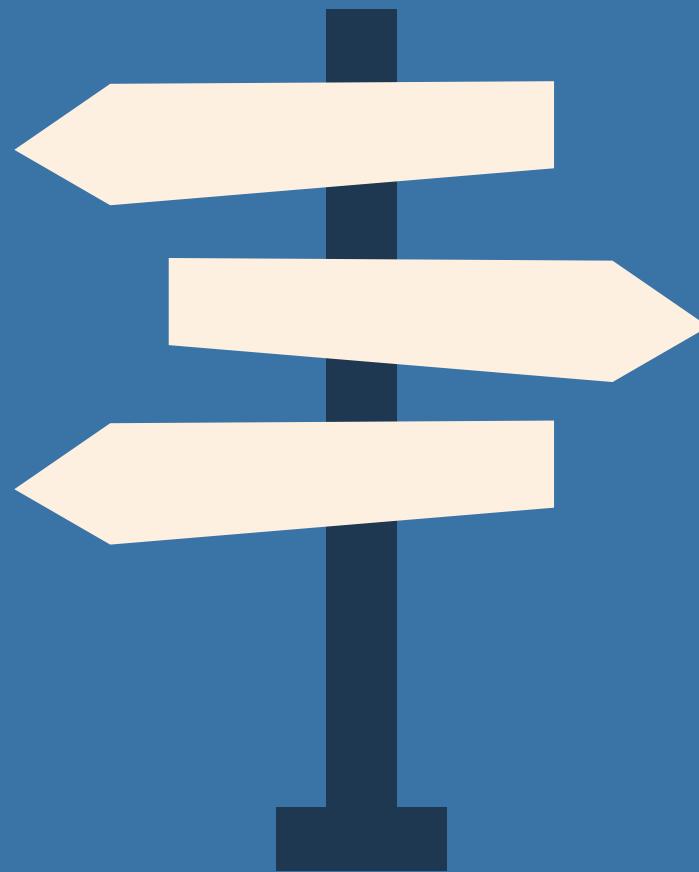
Note: You are 1 hour ahead of server time.

CHOICES

CHOICE TEXT	VOTES	DELETE?
<input type="text"/>	0	
<input type="text"/>	0	
<input type="text"/>	0	

+ Add another Choice

SAVE Save and add another Save and continue editing



Le routage explicite

■ ————— □
Chaque route est déclarée
manuellement dans un
fichier Python
□ ————— ■

■ ————— □
Ce type de routage
favorise la lisibilité et la
maintenance à grande
échelle
□ ————— ■

■ ————— □
Il renforce la séparation
des responsabilités :

- Le contrôleur logique (vue)
est dans `views.py`
- La déclaration des routes est
dans `urls.py`



Exemple

```
app_name = "polls"
urlpatterns = [
    path("", views.IndexView.as_view(), name="index"),
    path("<int:pk>/", views.DetailView.as_view(), name="detail"),
    path("<int:pk>/results/", views.ResultsView.as_view(), name="results"),
    path("<int:question_id>/vote/", views.vote, name="vote"),
]
```

urls.py

Maturité et communauté



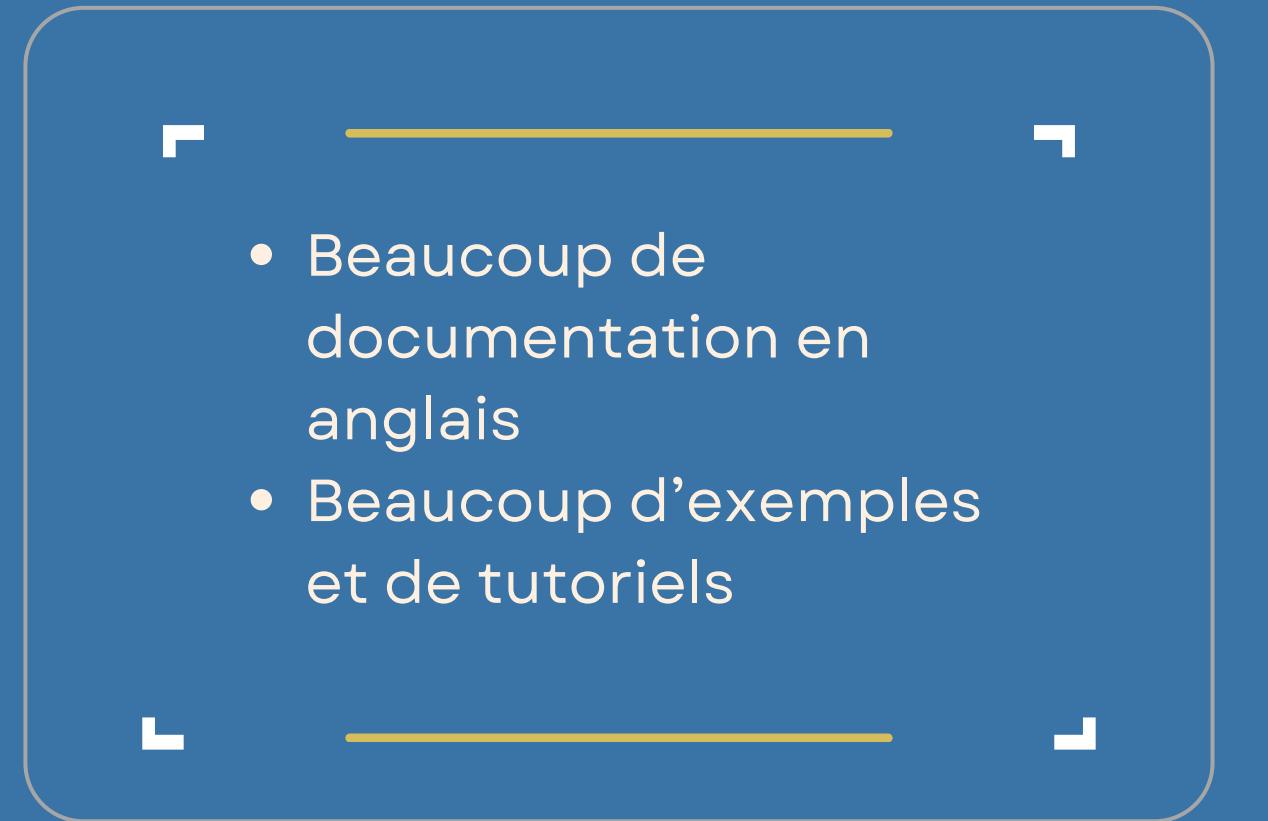
Communauté internationale

Plusieurs milliers de contributeurs

DjangoGirls

Conférence annuelle
DjangoCon





- Beaucoup de documentation en anglais
- Beaucoup d'exemples et de tutoriels

Documentation



Documentation officielle :

- <https://docs.djangoproject.com/>



- **1.0** : 3 septembre 2008
- **2.0** : 2 décembre 2017
- **3.0** : 2 décembre 2019
- **4.0** : 7 décembre 2021
- **5.0** : 4 décembre 2023
- **6.0** : décembre 2025



Maturité du code

Ecosystème de packages

Nom du package	Rôle principal
Celery	Exécution de tâches asynchrones
django-allauth	Authentification sociale (Google, GitHub, etc.)
django-crispy-forms	Formulaires HTML élégants
django-debug-toolbar	Débogage et analyse de requêtes SQL

Points forts



- ━━━━ ▶
Système d'authentification intégré

- ━━━━ ▶
Bonne gestion des exceptions et des backtraces

- ━━━━ ▶
Bonne documentation en anglais

- ━━━━ ▶
Prévention des attaques courantes de sécurité



Points faibles



- L'ORM de django peut causer des problèmes de performances

- Ne permet pas seul l'intégration d'AJAX côté client web



- Convient difficilement aux petits projets

- Manière standard de définir et d'exécuter les tâches

DJANGO - Points à retenir

Framework “Batteries incluses”

Besoins éditoriaux

Architecture MVT

Un projet = plusieurs apps

Admin généré, personnalisable et intégré au système d'authentification

Documentation et communauté

Adopté par de grandes entreprises

