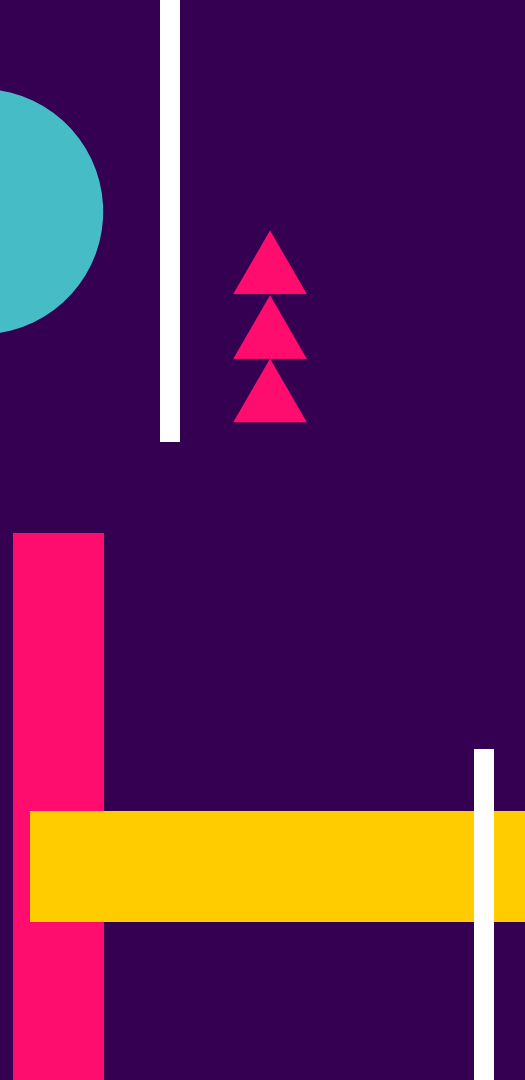
B3

La Merkanterie

Python - Django

1.

Objectifs



Développer avec python

- Développer une application web django

2.

Introduction

Django ?

- Framework de développement web open source
- Permet de développer des applications web rapidement

Pourquoi Django ?

- Sécurité intégrée
  - Fournit automatiquement la protection contre les attaques par injection de code et la gestion des utilisateurs et des autorisations
- Architecture MVT
  - Model view template
- ORM
  - Inclut un ORM (Object-Relational Mapping) puissant qui permet une interaction facile avec les bases de données et une gestion efficace des données.
- Gestion des formulaires facile
- URL Routing Dynamique
- Administration
  - inclut une interface d'administration par défaut qui permet aux développeurs de gérer facilement les données et les utilisateurs de l'application
- Grande communauté
- Flexibilité

3.

Installation

Installation

- Dans un terminal: pip3 install django
  - django-admin --version

Création d'un projet

- *django-admin startproject* + nom-du-projet
  - Création d'un répertoire avec le nom du projet
- Lancer le projet avec *python manage.py runserver*
- Ouvrez votre navigateur web et accédez à l'URL http://127.0.0.1:8000/ pour vérifier que votre projet fonctionne correctement.

Models

- Décrit les données de l’application
- Chaques modèle définit une table
- Les models sont définis dans le fichier models.py de chaques applications
- Apres modification d’un model il faut executer *python manage.py migrate*

```python
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    pub_date = models.DateTimeField(auto_now_add=True)

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

Vues

- Gère les requêtes HTTP
- Peuvent être définis en utilisant des fonctions ou des classes de vue
- Les vues sont définies dans le fichier views.py

```python
from django.shortcuts import render
from .models import Article

def article_list(request):
    articles = Article.objects.all()
    return render(request, 'articles/article_list.html', {'articles': articles})
```

Templates

- Fichier HTML qui définissent le contenu des pages web
- Séparation de la logique et la présentation de donnée
- Les templates sont stockés dans le répertoire templates

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Article list</title>
</head>
<body>
    <h1>Article list</h1>

    <ul>
    {% for article in articles %}
        <li>{{ article.title }} - {{ article.pub_date }}</li>
    {% endfor %}
    </ul>
</body>
</html>
```

Exemple

- django-admin startproject mysite
  - Python -m django startproject nom du projet pour les windows si django-admin not found
- manage.py est un script qui permet de gerer le projet
- __init__.py permet de considerer le repertoire comme un package
- setting.py est un fichier de configuration
- Urls.py définit les URL utilisé par l'app
- Asgi.py est un fichier de configuration qui définit l'interface de communication entre Django et un serveur ASGI (Asynchronous Server Gateway Interface).
- Wsgi.py est un fichier de configuration de Django qui définit l'interface de communication entre Django et un serveur WSGI (Web Server Gateway Interface).

Création d'une app de sondage

- python manage.py startapp polls

Vue

- polls/views.py

```python
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

URL de l'app

- polls/urls.py

```python
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

URL du projet

- mysite/urls.py
- http://localhost:8000/polls/

```python
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

Models

- polls/models.py

```python
import datetime

from django.db import models
from django.utils import timezone

# Create your models here.
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    def __str__(self):
        return self.question_text

    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

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

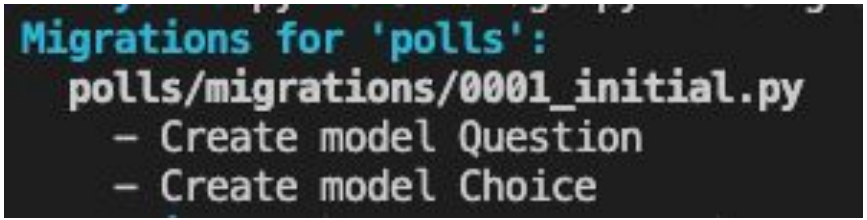
Référencement de l'app dans le projet

- mysite/settings.py

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Update des models

- python manage.py makemigrations polls



Création des tables en bdd

- python manage.py migrate

```
→  mysite python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying polls.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Django Admin

- Création d'un super user avec *python manage.py createsuperuser*
- http://127.0.0.1:8000/admin/

Ajout de l'app dans le back office

- polls/admin.py

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

# Register your models here.
admin.site.register(Question)
```

Formulaire

> polls > forms.py > QuestionForm

```python
from django import forms
from .models import Question

class QuestionForm(forms.ModelForm):

    class Meta:
        model = Question
        fields = ('question_text', 'pub_date')
```

Formulaire

> polls > views.py > question

```python
from django.shortcuts import render
from django.http import HttpResponse
from .models import Question
from .forms import QuestionForm



def question(request):
    if request.method == 'POST':
        form = QuestionForm(request.POST)
        if form.is_valid():
            form.save()
    else:
        form = QuestionForm()
    return render(request, 'polls/questions.html', {'form': form})
```

Formulaire

```
<form action="/polls/" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

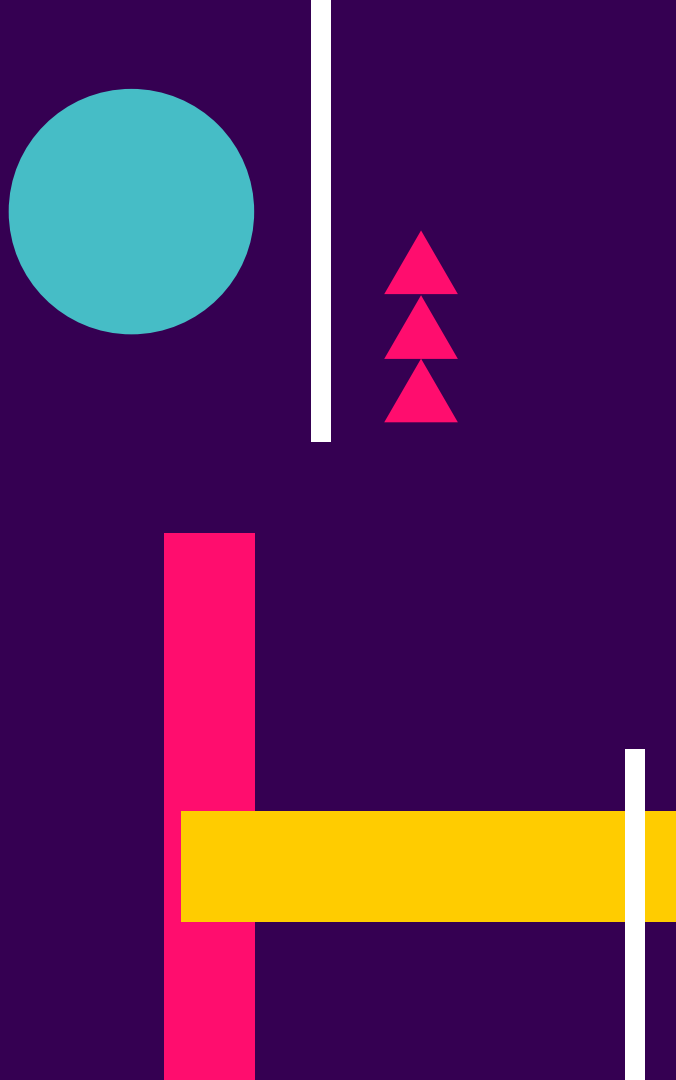
Cas pratique

App de notation de xxxx

- Créer une app capable de lister des xxxx
- Création d'une vue qui va permettre de voir les détails d'une xxxxx
- Création d'une vue qui donne la possibilité de noter les xxxx
- L'objet xxxxxxx a pour caractéristiques :
  - La marque
  - Le modèle
  - L'année de construction
  - La cylindrée
  - La version (break, coupé etc.)

4.

Debug

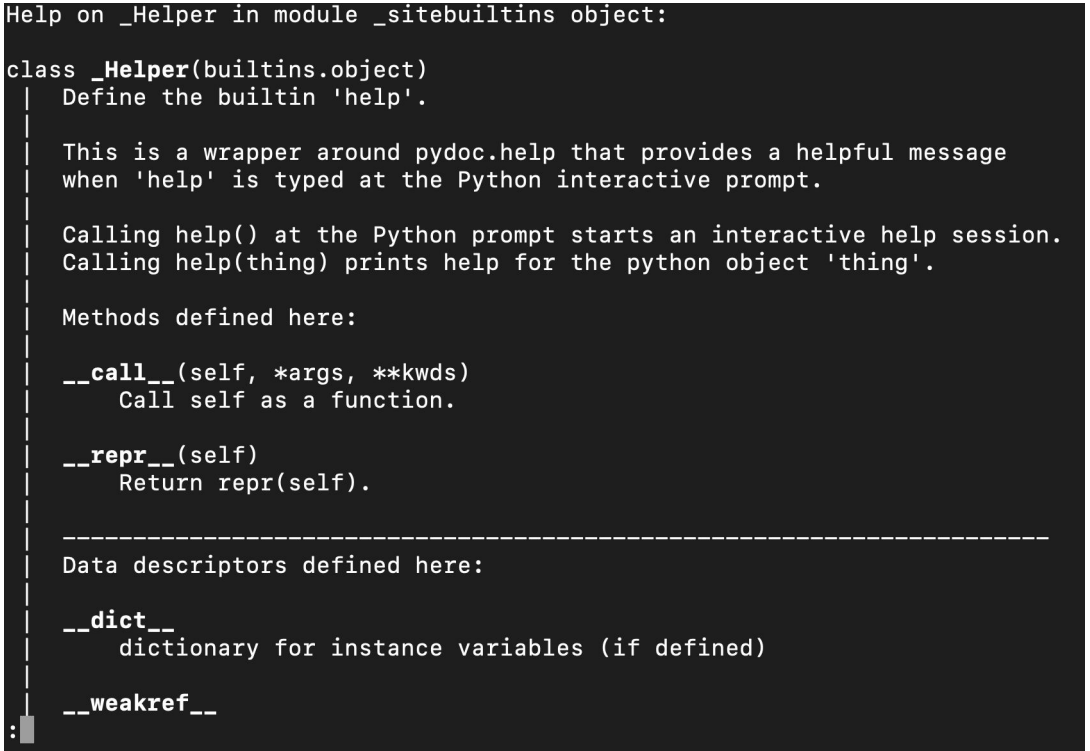


Python exceptions

- AttributError: Accés a un atribut ou a une méthode inexistante dans la classe
- KeyError: Appel a une clé inexistante de notre objet (liste, dictionnaire etc..)
- IndexError: On cherche la position d'un objet qui n'existe pas
- NameError: Appel a une variable inconnue
- SyntaxError
- TypeError
- ValueError
- FileNotFoundError
- ModuleNotFoundError

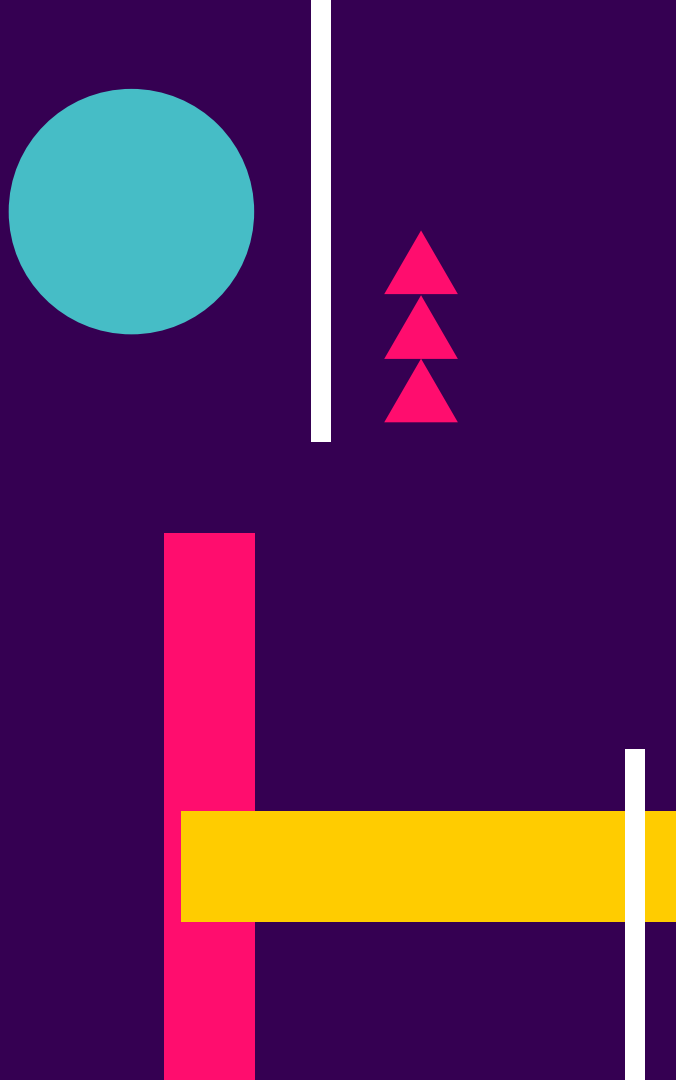
Help fonction

- Affiche le contenu des docstring de la classe ou de la fonction.



5.

User Model



AbstractUser ou AbstractBaseUser ?

- AbstractUser: Fournit tous les champs et fonctionnalités de la classe User

- AbstractBaseUser: obligation de fournir tous les champs voulut (sauf password qui est obligatoire)
  - Champs clé :
    - USERNAME_FIELD
    - EMAIL_FIELD
    - REQUIRED_FILED (liste des champs à spécifier lors de la création d'un SU)

Ressources

Logging

Doc

Python

Python basics

pdb

Doc