L’étude et les correctifs du code fourni.

Le constructeur des classes est manquant :

Les classes livre, dvd, cd, jeuDePlateau, et Emprunteur n'ont pas de constructeurs pour initialiser leurs attributs.
Pour corriger ce problème, il faut utiliser le constructeur **‘__init__’** afin de définir les attributs d'une instance de classe.

Correction de la classe « Livre » ;

```
class Livre:
    def __init__(self, name="", auteur="", date_emprunt="", disponible=True, emprunteur=""):
```

Remarque : « self » figurant dans la liste des arguments est l’entité désignant l’objet actif en cours de construction. À la suite de cet argument, des valeurs par défaut ont été incluses dans le constructeur.

Le nommage de certaines classes est incorrecte ;

Selon l’ensemble des règles de style de Python Extension Proposale (PEP8), les noms de classes doivent utiliser le CapWords convention.

Correction de la classe « Livre »:

```
class Livre:
```

Initialisation des variables d'instance ;

Les variables d'instance doivent être initialisées via le constructeur (__init__) pour chaque instance de la classe. Cela permet de définir les valeurs initiales des attributs lors de la création de l'objet.

Correction de la classe « Livre »:

```
class Livre:
    def __init__(self, name="", auteur="", date_emprunt="", disponible=True, emprunteur=""):
        self.name = name
        self.auteur = auteur
        self.date_emprunt = date_emprunt
        self.disponible = disponible
        self.emprunteur = emprunteur
```

La mise en place des fonctionnalités demandées.

- **Créer un membre-emprunteur :**

```
def add_emprunteur(request):
    # Si le formulaire a été soumis 'POST'.
    if request.method == 'POST':
        form = EmprunteurForm(request.POST)
        # Et si les données du formulaire sont valides.
        if form.is_valid():
            # Alors création d'un nouvel objet.
            form.save()
            # Puis redirection vers la vue 'list_emprunteurs'.
            return redirect('list_emprunteurs')
    else:
        # Sinon affichage d'un formulaire vide.
        form = EmprunteurForm()
    # Et redirection vers le template du formulaire.
    return render(request, 'librarian_member_list/member_add.html', {'form': form})
```

- **Afficher la liste des membres :**

```
def list_emprunteurs(request):
    # Requêtes pour récupérer tous les objets liés au modèle des emprunteurs depuis la base de données.
    emprunteurs = Emprunteur.objects.all()
    # Requêtes HTTP avec les objets récupérés.
    return render(request, 'librarian_member_list/member_list.html', {'emprunteurs': emprunteurs})
```

- **Mettre à jour un membre :**

```
def update_emprunteur(request, pk):
    # Récupération de l'objet Emprunteur correspondant à la clé primaire spécifiée.
    emprunteur = get_object_or_404(Emprunteur, pk=pk)
    if request.method == 'POST':
        # Si le formulaire a été soumis 'POST'.
        form = EmprunteurForm(request.POST, instance=emprunteur)
        if form.is_valid():
            # Alors création d'un nouvel objet.
            form.save()
            # Puis redirection vers la vue 'list_emprunteurs'.
            return redirect('list_emprunteurs')
    else:
        # Sinon, création d'un formulaire pré-rempli avec les données existant de l'objet "emprunteur" récupéré.
        form = EmprunteurForm(instance=emprunteur)
        # Et redirection vers le template du formulaire.
    return render(request, 'librarian_member_list/member_update.html', {'form': form, 'emprunteur': emprunteur})
```

- **Supprimer un membre :**

```
def delete_emprunteur(request, pk):
    # Récupération de l'objet Emprunteur correspondant à la clé primaire spécifiée.
    emprunteur = get_object_or_404(Emprunteur, pk=pk)
    # Si le formulaire a été soumis 'POST'.
    if request.method == 'POST':
        # Suppression de l'emprunteur.
        emprunteur.delete()
        # Puis redirection vers la vue 'list_emprunteurs'.
        return redirect('list_emprunteurs')
    # Sinon redirection vers le template du formulaire.
    return render(request, 'librarian_member_list/member_delete.html', {'emprunteur': emprunteur})
```

- **Afficher la liste des médias / Consultation de la liste des médias :**

```
def list_media(request):
    # Requêtes pour récupérer tous les objets depuis la base de données.
    livres = Livre.objects.all()
    dvds = DVD.objects.all()
    cds = CD.objects.all()
    boardgames = BoardGame.objects.all()

    # Requêtes HTTP avec les objets récupérés.
    return render(request, 'librarian_media_list/media_list.html', {
        'livres': livres,
        'dvds': dvds,
        'cds': cds,
        'boardgames': boardgames,

    })
```

- **Ajouter un media :**

```
def add_livre(request):
    # Si le formulaire a été soumis 'POST'.
    if request.method == 'POST':
        form = LivreForm(request.POST)
        # Et si les données du formulaire sont valides.
        if form.is_valid():
            # Alors création d'un nouvel objet.
            form.save()
            # Puis redirection vers la vue 'list_media'.
            return redirect('list_media')
    else:
        # Sinon affichage d'un formulaire vide.
        form = LivreForm()
        # Et redirection vers le template du formulaire.
    return render(request, 'librarian_media_list/mediaAdd/add_livre.html',
                  {'form': form})
```

- **Créer un emprunt pour un média disponible/ Rentrer un emprunt :**

```
def create_emprunt(request):
    # Si le formulaire a été soumis 'POST'.
    if request.method == 'POST':
        form = EmpruntForm(request.POST)
        # Et si les données du formulaire sont valides.
        if form.is_valid():
            # Alors, Crée une instance de l'emprunt à partir des données valides du
formulaire sans la sauvegarde en
            # base de données.
            emprunt = form.save(commit=False)
            # Si l'emprunteur a déjà 3 emprunts en cours.
            if Emprunt.objects.filter(emprunteur=emprunt.emprunteur,
date_retour__isnull=True).count() >= 3:
                # Alors, une erreur est ajoutée au formulaire.
                form.add_error(None, 'Un membre ne peut pas avoir plus de 3 emprunts à
la fois.')
            # Sinon, si l'emprunteur a un emprunt en retard.
            elif Emprunt.objects.filter(emprunteur=emprunt.emprunteur,
date_retour__isnull=True,
                                        date_emprunt__lt=timezone.now() -
timedelta(weeks=1)).exists():
                # Alors, une erreur est ajoutée au formulaire.
                form.add_error(None, 'Un membre ayant un emprunt en retard ne peut plus
emprunter.')
            # Sinon, le formulaire est validé et sauvegardé dans base de données.
            else:
                emprunt.save()
                # Et redirection vers le template du formulaire.
                return redirect('list_emprunteurs')
```

Stratégie de tests

Stratégie de tests pour les fonctionnalités permettant d'ajouté, supprimé, modifié une donnée de la base de données :

1) Testez les différentes vues en vérifiant que leur page s'affiche correctement (code de statut 200).
2) Vérifiez que les formulaires soient présents et correctement initialisé dans le contexte des vues.
3) Simulez une requête POST pour soumettre des données valides et vérifiez que la redirection après l'ajout est correcte (code de statut 302).
4) Vérifiez que l'objet a été ajouté, supprimé, modifié à la base de données avec les données soumises.

Exemple la fonctionnalité d'ajout de membres :

```python
import pytest
from django.test import Client
from django.urls import reverse
from librarian_app.forms import EmprunteurForm
from librarian_app.models import Emprunteur

@pytest.fixture
# Simulation de requêtes HTTP vers les vues.
def client():
    return Client()

@pytest.mark.django_db
def test_add_emprunteur_view(client):
    # Récupération de l'URL de la vue.
    url = reverse('add_emprunteur')
    # Effectue une requête vers l'URL  récupéré.
    response = client.get(url)
    # Vérifie que la réponse HTTP a un code de statut 200 (succès).
    assert response.status_code == 200
    # Vérifie que le formulaire 'form' est présent dans le contexte de la réponse.
    assert 'form' in response.context
    # Vérifie que le formulaire est correctement initialisé dans la vue.
    assert isinstance(response.context['form'], EmprunteurForm)
    # Simulation des données POST pour créer un nouveau membre .
    post_data = {
        'nom': 'Nouveau Nom de l\'Emprunteur',
        'bloquer': False,
    }
    # Effectue une requête POST vers l'URL de la vue en soumettant les données valides (post_data).
    response = client.post(url, post_data)
    # Vérifie que la réponse HTTP a un code de statut 302 (redirection).
    assert response.status_code == 302
    # Vérifie que la redirection se fait vers l'URL de 'list_emprunteurs' avec l'ajout de l'emprunteur réussi.
    assert response.url == reverse('list_emprunteurs')
    # Vérifie que l'objet Emprunteur a été ajouté à la base de données.
    assert Emprunteur.objects.filter(nom=post_data['nom']).exists()
```

Stratégie de tests pour les fonctionnalités permettant d'afficher les données :

1. Mettre en place des modèles avec des données de test.
2. Initialiser un client pour simuler des requêtes HTTP GET vers les vues d'affichage.
3. Vérifiez que la réponse HTTP a un code de statut 200.
4. Vérifiez que les objets de chaque type de média sont présents dans le contexte de la réponse.
5. Vérifiez que le bon template est utilisé pour rendre la vue.

Exemple la fonctionnalité d'affichage de la liste des médias :

```
import pytest
from django.test import Client
from django.urls import reverse
from librarian_app.models import Livre, DVD, CD, BoardGame

@pytest.mark.django_db
def test_list_media_view():
    # Création des modèles avec des données de test.
    Livre.objects.create(name='Livre 1', auteur='Auteur 1')
    DVD.objects.create(name='DVD 1', realisateur='Réalisateur 1')
    CD.objects.create(name='CD 1', artiste='Artiste 1')
    BoardGame.objects.create(name='Jeu de plateau 1', createur='Créateur 1')

    client = Client()
    url = reverse('list_media')

    response = client.get(url)

    # Vérifications
    assert response.status_code == 200
    # Vérifie que les clés sont présentes dans le contexte de la réponse et que donc les objets correspondants ont été passés au template.
    assert 'livres' in response.context
    assert 'dvds' in response.context
    assert 'cds' in response.context
    assert 'boardgames' in response.context

    # Vérifie que le bon template est utilisé.
    assert 'librarian_media_list/media_list.html' in [template.name for template in response.templates]
```

Instructions pour exécuter le programme.

Exécution du programme :

1. Ouvrir l'invite de commande.

2. Accéder au reptatoire du dossier « django project ».

3. Exécuter la commande python manage.py runserver.

Connexion au menu du bibliothécaire :

- username : librarian
- Password : Django

URL du menu : librarian_app/menu/