Année 2024/2025

Rapport de projet

Mini projet de programmation en Python

Système de Gestion de Bibliothèque

Projet réalisé par :

Bouazza Hafsa

Professeur :

Haja Zakaria

Filière :

Génie Informatique

Introduction :

Dans le cadre de l'apprentissage du langage Python orienté objet, ce mini-projet consiste à développer une **application de gestion d'une bibliothèque**.
Le projet met en œuvre plusieurs notions clés : la programmation orientée objet, la gestion de fichiers, les interfaces graphiques avec Tkinter et la visualisation de données avec Matplotlib.

L'application permet de gérer efficacement un catalogue de livres, les membres de la bibliothèque, ainsi que les opérations d'emprunt et de retour.
Elle offre également des **statistiques graphiques** pour analyser l'activité de la bibliothèque, telles que la répartition des livres par genre ou le suivi des emprunts.

Ce projet constitue une **mise en pratique complète** des compétences acquises en Python, en abordant aussi bien la logique métier que la création d'interfaces conviviales pour l'utilisateur.

Présentation du projet

Ce projet est une application de gestion de bibliothèque développée en **Python**.
Elle permet de gérer les livres disponibles, les membres inscrits, ainsi que les opérations d'emprunt et de retour.

L'application dispose d'une **interface graphique intuitive** réalisée avec **Tkinter**, facilitant l'interaction avec l'utilisateur.

Le projet est structuré autour de **trois entités principales** :

- **Les livres**, contenant des informations telles que le titre, l'auteur, le genre, etc.
- **Les membres**, identifiés par un ID et un nom.
- **La bibliothèque**, qui centralise la gestion des emprunts et l'ensemble des données.

Cet outil s'adresse principalement aux **petites bibliothèques** (écoles, associations, etc.) ou aux **étudiants souhaitant apprendre** la programmation orientée objet et la création d'interfaces graphiques.

Objectifs du Projet

L'objectif principal est de développer une application complète de gestion de bibliothèque en **langage Python**, respectant les principes de la **programmation orientée objet (POO)**.

Plus précisément, ce projet vise à :

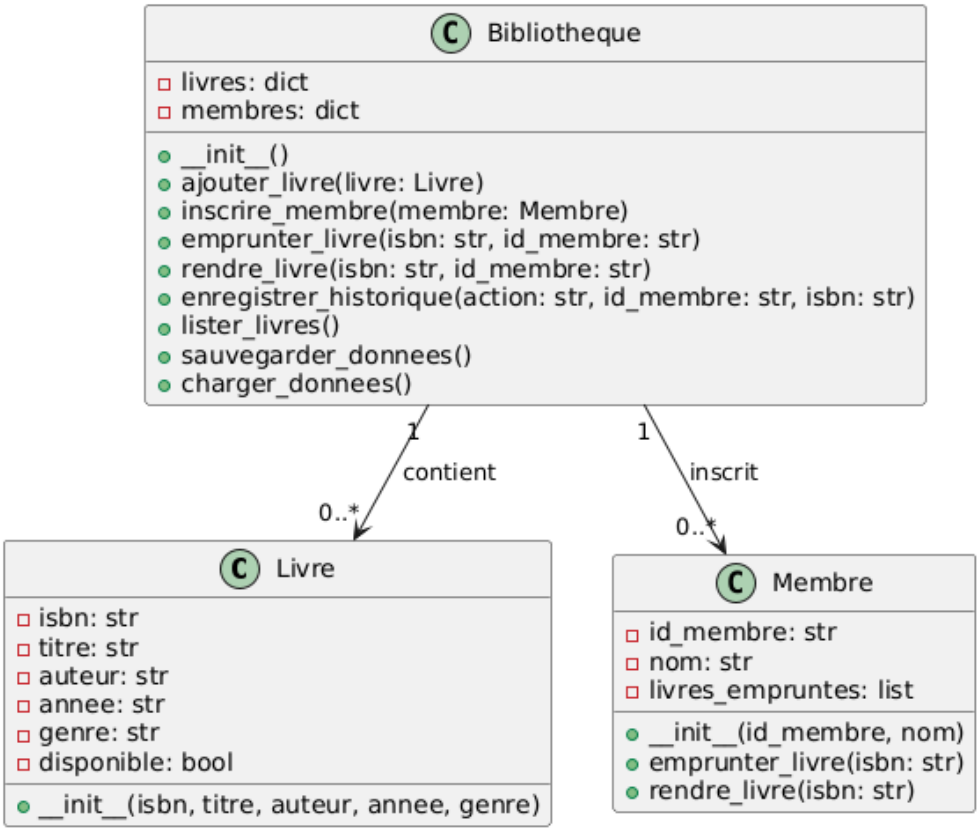
- Gérer un **catalogue de livres** (ajout, modification, suppression, disponibilité).
- Gérer les **membres inscrits** à la bibliothèque.
- Gérer les **emprunts et retours**, avec un suivi via un fichier **historique**.
- Offrir une **interface graphique conviviale** (Tkinter) pour les utilisateurs.
- Générer des **statistiques visuelles** (diagramme circulaire, histogramme, courbe) avec **Matplotlib**.
- Assurer la **sauvegarde persistante** des données dans des fichiers **TXT** et **CSV**.

Ce projet permet de mobiliser plusieurs compétences essentielles :

- La structuration du code en **classes et modules**.
- La **gestion d'événements** via une interface graphique.
- La **manipulation de fichiers** pour la persistance des données.
- L'utilisation de **bibliothèques Python externes** (Tkinter, Matplotlib).

Diagramme de classes UML :

1. Représentation graphique



2. Description des classes

◆ ***Classe Livre***

Cette classe représente un livre dans la bibliothèque. Elle contient des attributs comme le titre, l'auteur, l'année, le genre, et un indicateur de disponibilité. Elle permet d'instancier les livres à partir de leurs caractéristiques de base.

◆ ***Classe Membre***

Elle modélise un membre inscrit dans la bibliothèque. Chaque membre possède un identifiant, un nom, ainsi qu'une liste des livres empruntés. Elle offre des méthodes pour emprunter et rendre un livre.

◆ ***Classe Bibliotheque***

Il s'agit de la classe centrale du système. Elle contient l'ensemble des livres et des membres, et assure la gestion des opérations principales : ajouter un livre, inscrire un membre, emprunter ou rendre un livre, sauvegarder les données et enregistrer l'historique. Elle sert d'interface entre l'utilisateur et les autres entités du système.

Explication des algorithmes clés

1. Algorithmes de gestion (dans bibliotheque.py)

Ce module constitue le **noyau fonctionnel** de l'application. Il contient la logique de gestion des livres, des membres et des emprunts. Les classes sont conçues en respectant les principes de la **programmation orientée objet**, assurant modularité et réutilisabilité.

● ***emprunter_livre() – Gestion d'un emprunt :***

Cette méthode permet à un membre d'emprunter un livre en vérifiant plusieurs conditions :

1. Vérifie si le livre existe dans le catalogue.

2. Vérifie si le membre est bien inscrit.

3. Vérifie que le livre est disponible.

4. Vérifie que le membre n'a pas déjà atteint sa limite de 3 livres.

5. Marque le livre comme emprunté (non disponible).

6. Ajoute l'ISBN du livre dans la liste des livres empruntés par le membre.

7. Enregistre l'action dans le fichier `historique.csv`.

Elle déclenche des exceptions personnalisées (`LivreInexistantError`, `MembreInexistantError`, `LivreIndisponibleError`, `QuotaEmpruntDepasseError`) si une des conditions échoue.

● ***rendre_livre() – Retour d'un livre :***

Cette méthode permet à un membre de rendre un livre :

1. Vérifie que le livre et le membre existent.

2. Vérifie que le livre a bien été emprunté par ce membre.

3. Rend le livre disponible à nouveau.

4. Supprime l'ISBN du livre de la liste du membre.

5. Enregistre l'action de retour dans l'historique.

Cette méthode assure également l'intégrité du retour : un membre ne peut pas rendre un livre qu'il n'a pas emprunté.

● ***sauvegarder_donnees() – Sauvegarde des données :***

Cette méthode enregistre l'état actuel de la bibliothèque dans deux fichiers texte :

- `livres.txt` : contient les données des livres (isbn, titre, auteur, etc.)

- `membres.txt` : contient les données des membres et leurs livres empruntés

L'objectif est de rendre les données persistantes entre les sessions de l'application.

● ***enregistrer_historique() – Suivi des emprunts/retours :***

Cette méthode écrit dans le fichier `historique.csv` chaque action d'emprunt ou de retour, avec :

- La date et l'heure

- L'action effectuée

- L'ID du membre

- L'ISBN du livre concerné

Elle est utilisée par les méthodes `emprunter_livre()` et `rendre_livre()` pour permettre une traçabilité complète.

2. Algorithmes de visualisation (dans visualisations.py)

Ce module regroupe plusieurs fonctions permettant de représenter graphiquement les données de la bibliothèque. Ces visualisations sont utiles pour analyser l'utilisation des ressources et fournir des insights visuels à l'utilisateur. Voici les algorithmes clés utilisés :

● *diagramme_genres()*

But : Afficher un diagramme circulaire représentant la répartition des livres par genre.
Principe :

1. Lecture du fichier livres.txt.
2. Extraction du genre de chaque livre.
3. Comptage de l'occurrence de chaque genre à l'aide de Counter.
4. Affichage d'un diagramme circulaire (plt.pie) avec les pourcentages.

Utilité : Identifier visuellement quels genres sont les plus représentés dans la bibliothèque.

● *top_auteurs()*

But : Afficher un histogramme des 10 auteurs les plus fréquents dans le catalogue.
Principe :

1. Lecture du fichier livres.txt.
2. Extraction des noms d'auteurs.
3. Comptage avec Counter, puis sélection des 10 premiers avec most_common(10).
4. Affichage via plt.bar.

Utilité : Mettre en évidence les auteurs les plus populaires dans la collection.

● *activite_emprunts()*

But : Afficher une courbe illustrant le nombre d'emprunts effectués sur les 30 derniers jours.
Principe :

1. Lecture du fichier historique.csv.
2. Filtrage des lignes contenant l'action "emprunt".
3. Extraction et conversion de la date d'emprunt.
4. Création d'un tableau contenant les 30 derniers jours.
5. Comptage du nombre d'emprunts par jour.
6. Affichage de la courbe avec plt.plot.

Utilité : Suivre l'évolution de l'activité d'emprunt au fil du temps.

● ***livres_par_genre(biblio)***

But : Afficher une répartition des livres par genre à partir de l'objet Bibliotheque.
Principe :

1. Parcours des livres enregistrés dans l'objet biblio.
2. Extraction du genre pour chaque livre.
3. Comptage avec Counter.
4. Affichage d'un diagramme circulaire.

Différence avec diagramme_genres() :
Cette version fonctionne **directement avec les objets** en mémoire, sans relecture du fichier livres.txt.

● ***emprunts_par_membre()***

But : Afficher un histogramme du nombre d'emprunts effectués par chaque membre.
Principe :

1. Lecture du fichier historique.csv.
2. Filtrage des actions "emprunt".
3. Comptage du nombre d'emprunts par id_membre.
4. Affichage d'un histogramme avec plt.bar.

Utilité : Identifier les membres les plus actifs.

3. Algorithmes de l'interface utilisateur (interface.py)

L'interface graphique de l'application est construite à l'aide de la bibliothèque **Tkinter**, permettant une interaction intuitive avec les fonctionnalités du système. Elle est structurée en plusieurs **onglets** correspondant aux actions principales : gestion des livres, des membres, des emprunts et des statistiques.

Structure générale :

L'interface utilise le widget ttk.Notebook pour créer des onglets séparés. Chaque onglet contient :

- Des champs d'entrée (Entry, Combobox) pour ajouter/modifier des données.
- Des boutons (Button) associés à des **fonctions de rappel (command=...)**.
- Des tableaux (Treeview) pour afficher les données chargées depuis la classe Bibliotheque.

● ***Ajouter un livre / membre***

Fonctions associées :

- ajouter_livre()
- ajouter_membre()

Principe :

1. Récupération des données saisies par l'utilisateur via Entry.
2. Création d'un objet Livre ou Membre.
3. Appel de la méthode de la classe Bibliotheque correspondante.
4. Mise à jour du tableau (Treeview) et sauvegarde.

● *Emprunter / Rendre un livre*

Fonctions associées :

- emprunter_livre()
- rendre_livre()

Principe :

1. Lecture de l'ISBN et de l'ID du membre depuis l'interface.
2. Appel des méthodes emprunter_livre() ou rendre_livre() de Bibliotheque.
3. Gestion des erreurs avec try/except pour afficher des messages informatifs.
4. Mise à jour de l'affichage.

● *Affichage des statistiques*

Fonctions associées à chaque bouton :

- visualisations.diagramme_genres()
- visualisations.top_auteurs()
- visualisations.activite_emprunts()
- visualisations.emprunts_par_membre()
- visualisations.livres_par_genre(biblio)

Principe :
Chaque bouton dans l'onglet **Statistiques** appelle une fonction du module visualisations.py. Ces fonctions utilisent **Matplotlib** pour générer et afficher des graphiques basés sur les fichiers livres.txt et historique.csv ou l'objet biblio.

● *Sauvegarde et chargement automatiques*

À chaque ajout/modification d'un livre ou membre, l'appel à sauvegarder_donnees() permet de :

- Écrire les livres dans data/livres.txt
- Écrire les membres dans data/membres.txt

Lors du lancement de l'interface, charger_donnees() est exécutée pour recharger toutes les données stockées.

Visualisations :

Ce module a pour objectif de fournir une représentation graphique claire des données manipulées par l'application, afin d'aider les utilisateurs à mieux comprendre l'état de la bibliothèque.

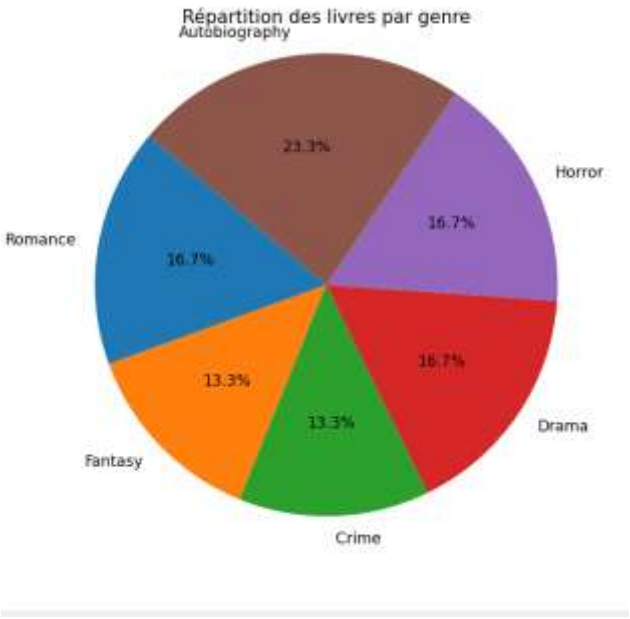
Grâce à la bibliothèque **Matplotlib**, plusieurs types de graphiques sont générés automatiquement à partir des fichiers de données (livres.txt et historique.csv). Ces visualisations permettent d'avoir une vue d'ensemble rapide et intuitive sur :

- La répartition des livres selon leur genre.
- Les auteurs les plus présents dans le catalogue.
- L'évolution des emprunts dans le temps.
- L'activité des membres (nombre d'emprunts par utilisateur).

Les graphiques sont accessibles directement depuis l'onglet **Statistiques** de l'interface Tkinter, et sont mis à jour en temps réel selon les données enregistrées.

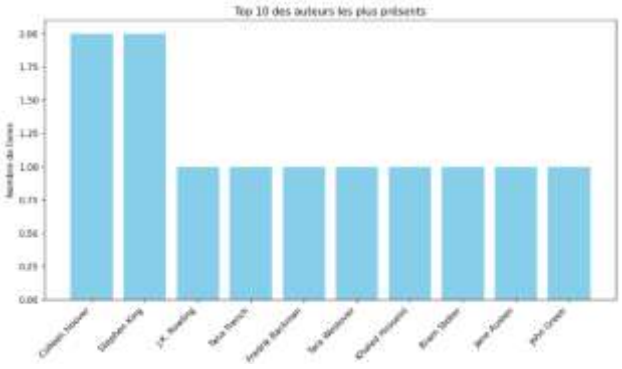
1. Diagramme circulaire : % de livres par genre

Ce **diagramme circulaire** illustre la proportion de chaque **genre littéraire** au sein de la bibliothèque.
Il est généré à partir des données du fichier livres.txt, en analysant la colonne correspondant au genre (genre).

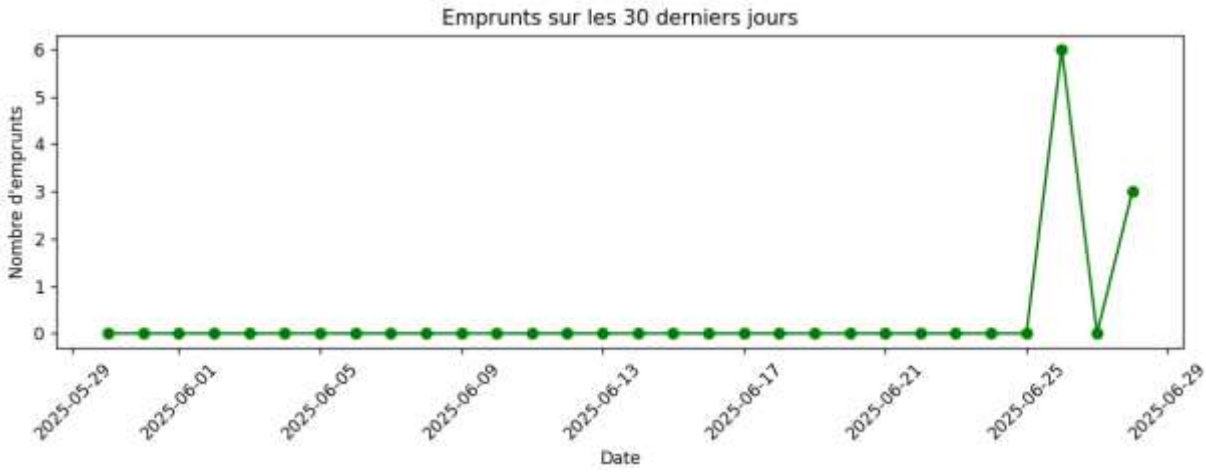


2. Histogramme : Top 10 des auteurs les plus populaires

Ce **diagramme en barres** représente les **10 auteurs les plus présents** dans le catalogue de la bibliothèque.
Il est généré à partir des données du fichier livres.txt, en comptant la fréquence de chaque auteur (auteur).



3. Courbe temporelle : activité des emprunts (30 derniers jours)



Cette **courbe temporelle** affiche le **nombre quotidien d'emprunts** sur une période de 30 jours. Elle s'appuie sur les données enregistrées dans historique.csv, en filtrant uniquement les lignes où l'action est "emprunt".

Difficultés rencontrées et solutions :

1. Problème de gestion des fichiers et des chemins relatifs

Difficulté :
Lors de l'accès aux fichiers comme livres.txt ou historique.csv, des erreurs de type FileNotFoundError survenaient, surtout lors de l'exécution depuis différents répertoires.

Solution :
Utilisation des fonctions os.path.dirname() et os.path.join() pour construire des chemins **relatifs dynamiques**, indépendants du répertoire courant d'exécution.

2. Sauvegarde et chargement des données

Difficulté :
Assurer la **persistance des données** (livres, membres, historique) entre les exécutions sans base de données.

Solution :
Utilisation des formats **TXT** et **CSV** :

- livres.txt pour les livres,
- membres.txt pour les membres,
- historique.csv pour tracer les emprunts/retours.

Des méthodes spécifiques ont été codées dans la classe Bibliotheque pour automatiser la lecture/écriture de ces fichiers.

3. Visualisation avec Matplotlib dans une interface Tkinter

Difficulté :
Afficher les visualisations Matplotlib à partir de boutons Tkinter, tout en gardant l'interface réactive.

Solution :
Les fonctions de visualisation sont séparées dans un module dédié visualisations.py, et appelées via les command= des boutons.
Les graphiques s'affichent dans une nouvelle fenêtre indépendante grâce à plt.show().

4. Gestion des erreurs métiers

Difficulté :
Il fallait éviter les emprunts de livres inexistants, indisponibles ou par des membres non inscrits.

Solution :
Création d'un module exceptions.py regroupant des **exceptions personnalisées** (comme LivreInexistantError, MembreInexistantError, etc.) utilisées dans bibliotheque.py avec try/except.

Conclusion

Ce projet de gestion de bibliothèque a permis de mettre en œuvre un ensemble complet de compétences en programmation orientée objet, en manipulation de fichiers, en interface graphique avec Tkinter, ainsi qu'en visualisation de données avec Matplotlib.

L'application obtenue est fonctionnelle, intuitive et adaptée à une petite bibliothèque, que ce soit dans un cadre éducatif ou associatif. Elle permet de gérer efficacement les livres, les membres et les emprunts, tout en offrant des outils d'analyse visuelle utiles à travers des statistiques graphiques.

Ce travail a également été l'occasion de faire face à des défis techniques, tels que la persistance des données, la gestion des erreurs ou l'intégration graphique, et d'y apporter des solutions concrètes et structurées.

En somme, ce projet constitue une base solide pour toute personne souhaitant approfondir ses connaissances en Python et en conception logicielle.