Exercice 1 : Gestion d'une bibliothèque

Vous allez créer une application pour gérer une bibliothèque. Cette application doit permettre de gérer des livres, des utilisateurs et les emprunts. Vous devez implémenter les concepts suivants :

1. Classe et Objets

- Créez une classe `Livre` avec les attributs suivants :
  - `titre` (le titre du livre),
  - `auteur` (le nom de l'auteur),
  - `annee_publication` (l'année de publication),
  - `disponible` (booléen, indique si le livre est disponible ou non).
- Ajoutez une méthode `afficher_details()` pour afficher les détails du livre.

2. Encapsulation

- **Déclarer les champs private**

- Ajoutez des getters et setters pour accéder et modifier les attributs du livre (par exemple, pour marquer un livre comme emprunté ou retourné).

3. Héritage

- Créez une classe `Utilisateur` avec les attributs suivants :
  - `nom` (nom de l'utilisateur),
  - `email` (email de l'utilisateur),
  - `livres_empruntes` (une liste des livres empruntés par cet utilisateur).
- Créez une classe `Bibliothecaire`, qui hérite de `Utilisateur`, avec une méthode supplémentaire `ajouter_livre()` pour ajouter un livre à la bibliothèque.

4. Polymorphisme

- Ajoutez une méthode `afficher_details()` dans la classe `Utilisateur` qui affiche les informations de l'utilisateur et les livres qu'il a empruntés.
- La méthode `afficher_details()` doit être redéfinie dans la classe `Bibliothecaire` pour afficher également un message indiquant que l'utilisateur est un bibliothécaire.

5. Abstraction

- Créez une classe abstraite `Personne` avec une méthode abstraite `afficher_details()`.
- Les classes `Utilisateur` et `Bibliothecaire` doivent hériter de cette classe et implémenter la méthode `afficher_details()`.

Exercice 2 : Gestion d'une bibliothèque (Version avec Méthodes Magiques)

Vous allez créer une application pour gérer une bibliothèque en utilisant les méthodes magiques. Voici les spécifications :

Classes et Concepts

1. **Classe `Livre` :**
   - Utilisez `__str__` pour afficher les détails du livre sous forme de texte.
   - Implémentez `__eq__` pour comparer deux livres (par exemple, deux livres sont égaux s'ils ont le même titre et le même auteur).
   - Implémentez `__lt__` et `__gt__` pour comparer les livres en fonction de leur année de publication.
2. **Classe `Bibliotheque` :**
   - Utilisez `__len__` pour retourner le nombre total de livres dans la bibliothèque.
   - Utilisez `__getitem__` pour accéder à un livre via son index.
   - Implémentez `__contains__` pour vérifier si un livre est dans la bibliothèque.
3. **Classe `Utilisateur` :**
   - Implémentez `__repr__` pour afficher les informations de l'utilisateur et les livres empruntés.
   - Utilisez `__call__` pour permettre à un utilisateur d'emprunter un livre en utilisant une syntaxe simplifiée.

Exercice3 : Gestion des adresses IPv4

Vous devez implémenter un système de gestion des adresses IPv4 en POO avec les spécifications suivantes :

Classe abstraite Adresse :

- Cette classe représente une adresse générique.
- Elle contient des méthodes abstraites que les classes dérivées doivent implémenter :
  - valider(): Vérifie si l'adresse est valide.
  - get_type(): Retourne le type de l'adresse (ex. IPv4, IPv6).
- Contient une méthode spéciale __str__ pour retourner l'adresse sous forme de chaîne de caractères.
- Contient une méthode spéciale __eq__ pour vérifier si deux adresses sont égales.

2. **Classe IPv4 :**
   - Hérite de la classe abstraite Adresse.
   - Représente une adresse IPv4.
   - Attributs :
     - octets: Une liste contenant 4 entiers représentant les octets de l'adresse (ex. [192, 168, 1, 1]).
   - Constructeur par défaut : Initialise l'adresse à 0.0.0.0.
   - Constructeur personnalisé : Permet d'initialiser l'adresse avec une liste d'octets.

- Implémente la méthode valider() :
  - Vérifie que chaque octet est compris entre 0 et 255.
- Implémente la méthode get_type() : Retourne "IPv4".
- Méthode spéciale __contains__ :
  - Permet de vérifier si un octet donné est présent dans l'adresse (ex. 192 in ipv4 retourne True si l'octet 192 est présent).
- Méthode spéciale __str__ :
  - Retourne l'adresse sous forme de chaîne (ex. "192.168.1.1").
- Méthode spéciale __eq__ :
  - Compare deux objets IPv4 pour vérifier si les adresses sont identiques.

3. **Classe ReseauIPv4 :**
   - Représente un réseau IPv4.
   - Attributs :
     - adresse: Une instance de la classe IPv4 représentant l'adresse du réseau.
     - masque: Une instance de la classe IPv4 représentant le masque de sous-réseau.
   - Méthodes :
     - valider_reseau(): Vérifie que l'adresse du réseau et le masque sont valides.
     - __str__: Retourne une représentation sous forme de chaîne (ex. "192.168.1.0 / 255.255.255.0").
     - __contains__: Permet de vérifier si une adresse IPv4 donnée appartient au réseau.

Exercice 4 : TKinter