TP SYNTHESE :

Créer un répertoire Tp_synthèse (dans le répertoire git déjà partagé) dans lequel vous aller créer 4 sous-répertoires tp1, tp2, tp3 et tp4. Mettre le code de chaque Tp dans le répertoire correspondant.
Le code conçu sera à envoyer au plutard à 17h00

TP1 : console et BDD

Client - CompteBancaire - Agence
Client (idClient, Civilite, nom, prénom, adresse)
CompteBancaire(idCompte, solde, decouvert) exemple le compte (4, 500, 200) à un solde de 500€ et à 200€ de découvert autorisé
Agence(idAgence, nomAgence)

Un client possède plusieurs compte Bancaires
Une agence gère plusieurs comptes bancaires.
1 - Faire le CRUD de Client, CompteBancaire et Agence
2 - Créer les méthodes ajouter(client, compte) qui permet à un client d'ajouter de l'argent dans son compte
3 - Créer la méthode retirer(qui permet à un client de retirer de l'argent dans son compte en respectant le découvert autorisé
4 - Créer la méthode listerCompteAgence(agence) qui liste les comptes d'une agence donnée
5 - Créer la méthode listerComptes(client) qui liste les comptes d'un client donné
6- Dans le programme principal, testez toutes les méthodes réalisées dans les questions précédentes.

TP2: Classes et Fichiers

Considérons les deux classes Personne et Adresse. Les attributs de la classe Adresse sont :
_rue : un attribut privé de type chaîne de caractère.
_ville : un attribut privé de type chaîne de caractère.
_code postal : un attribut privé de type chaîne de caractère.
Les attributs de la classe Personne sont :
_nom : un attribut privé de type chaîne de caractère.
_sexe : un attribut privé de type chaîne de caractère (cet attribut aura comme valeur soit 'M' soit 'F').
_adresses : un attribut privé de type tableau d'objet de la classe Adresse.

1. Créez les deux classes Adresse et Personne dans deux fichiers différents. N'oubliez pas de d´en définir les getters/setters et les constructeurs.

2. Créez une troisième classe ListePersonnes ayant un seul attribut personnes : un tableau d'objets Personne. Définissez les getters/setters et le constructeur de cette classe.

3. Ecrivez la méthode ´ find_by_nom(s: str) qui permet de chercher dans le tableau personnes si l'attribut nom d'un est ´egal `a la valeur du paramètre s. Si c'est le cas, elle retourne le premier objet correspondant, sinon null.

4. Ecrivez la méthode ´ exists_code_postal(cp: str) qui permet de vérifier dans le tableau personnes si un objet possède au moins une adresse dont le code postal est égal au paramètre cp.
Si c'est le cas, elle retourne True, sinon False.

5. Ecrivez la méthode count_personne_ville(ville: str) qui permet de calculer le nombre d'objets dans le tableau personnes ayant une adresse dans la ville passée en paramètre.

6. Ecrivez la méthode ´ edit_personne_nom(oldNom: str, newNom: str) qui remplace les noms de personnes ayant un nom égal à la valeur oldNom par newNom

7. Ecrivez la méthode edit_personne_ville(nom: str, newVille: str) qui remplace les villes de personnes ayant un nom egal à la valeur du paramètre nom par newVille

8. Écrire la méthode write_to_file(fichier:str) qui va parcourir le tableau d'objets de personne et écrit lign
e par ligne chaque objet Personne

9. Écrire la méthode read_from_file(fichier:str) qui va parcourir chaque ligne du fichier passé en paramètre et retourne un tableau d'objet Personne
10. Dans le programme principal, testez toutes les méthodes réalisées dans les questions précédentes.

TP3 : Héritage

Créer les héritages suivants:
- La sous-classe AGrave hérite de trois classes A, Accent et Abracadabra dans cet ordre
- La classe A hérite elle même de la classe Alphabet
- La Abracadabra hérite de Mot

Alphabet a pour attribut mot
Les autres classes n'ont pas d'attribut

La classe Alphabet a pour méthodes : info(self), test1(self), test2(self)
La classe Mot a pour méthodes :info(self), test1(self), test3(self)
La classe Accent a pour méthodes : info(self), test2(self), test3(self)
La classe A a pour méthodes : info(self), test1(self)
La classe Abracadabra a pour méthodes : test1(self)
La classe AGrave a pour méthodes :test4()
Les méthodes sont juste des affichages (exemple de la classe Alphabet ci-dessous)

```
class Alphabet:
    def __init__(self, nom):
        self.lettre_nom = nom
    def info(self):
        print("Je suis une lettre de l'alphabet")
    def test1(self):
        print("Fonction test1() de la classe Alphabet")
    def test2(self):
        print("Fonction test2() de la classe Alphabet")
```

Tester les appels suivants:
- aAccentGrave = AGrave("à")
- afficher aAccentGrave.nom
- aAccentGrave.test1()
- aAccentGrave.test2()
- aAccentGrave.test3()
- aAccentGrave.test4()

Donner une conclusion concernant l'appel des méthodes dans le cadre d'un héritage multiple

TP4: tkinter et BDD (optionnel)

Créer un formulaire tkinter avec un champ de saisie login et un champ de saisie password.
Rajouter un bouton connexion.
Créer une base de données sqlite tt_users contenant une table personne (id, login, password)
Lorsqu'on clique sur le bouton connexion, si l'utilisateur existe afficher un popup "utilisateur connecté " sinon afficher dans le popup " erreur de connexion "