DS : Requêtes SQL / Dictionnaires

1 Exercice 1

On souhaite créer un programme permettant de gérer un **carnet d'adresses** à l'aide de la structure de données `dictionnaire` en Python.

Chaque contact sera représenté par un **dictionnaire** contenant les informations suivantes :

- `telephone`
- `email`
- `ville`

L'ensemble des contacts sera stocké dans un **dictionnaire principal**, dont les clés seront les noms (ou identifiants) des contacts.

Exemple :

```
contacts = {
    "Alice": {"telephone": "0612345678",
     "email": "alice@mail.com", "ville": "Paris"},
    "Bob": {"telephone": "0755667788", "email": "bob@mail.com",
     "ville": "Lyon"}}
```

Vous devez rédiger les **six fonctions suivantes** permettant de manipuler ce carnet d'adresses.

1. Fonction `ajouter_contact(contacts, nom, telephone, email, ville)`
   - Ajoute un nouveau contact dans le dictionnaire `contacts`.
   - Si le contact existe déjà, la fonction ne doit **pas l'écraser** et doit afficher un message d'avertissement.
   - Retourne le dictionnaire mis à jour.
2. Fonction `supprimer_contact(contacts, nom)`
   - Supprime le contact portant le nom indiqué, s'il existe.
   - Si le nom n'est pas trouvé, affiche un message d'erreur.
   - Retourne le dictionnaire mis à jour.
3. Fonction `rechercher_contact(contacts, nom)`
   - Recherche un contact par son nom.
   - Si trouvé, retourne le dictionnaire associé à ce contact.
   - Sinon, retourne `None`.
4. Fonction `modifier_contact(contacts, nom, champ, nouvelle_valeur)`
   - Permet de modifier un champ (`telephone`, `email` ou `ville`) pour un contact donné.

- Si le contact ou le champ n'existe pas, affiche un message d'erreur.
- Retourne le dictionnaire mis à jour.

5. Fonction `afficher_contacts(contacts)`
   - Affiche proprement la liste de tous les contacts sous la forme :

     ```
     Nom : Alice | Téléphone : 0612345678 | Email : alice@mail.com
      | Ville : Paris
     ```

   - Si le dictionnaire est vide, indique que le carnet est vide.
6. Fonction `rechercher_par_ville(contacts, ville)`
   - Retourne une **liste** des noms des contacts habitant dans la ville donnée.
   - Si aucun contact ne correspond, retourne une liste vide.
7. Fonction `hachage_email(contacts, nom)`
   - Calcule une valeur de hachage simple de l'adresse email d'un contact.
   - Le hachage est obtenu en additionnant les codes ASCII de tous les caractères de l'email.
   - Si le contact n'existe pas, la fonction doit afficher un message d'erreur.
   - Retourne la valeur entière correspondant au hachage.

   Rappel : Le **code ASCII** associe à chaque caractère un nombre entier (compris entre 0 et 127), appelé son *code ASCII*.

   Par exemple : `'A'` $\rightarrow$ 65, `'a'` $\rightarrow$ 97, `'@'` $\rightarrow$ 64.

Exemple d'utilisation attendue

```
contacts = {}
contacts = ajouter_contact(contacts,
 "Alice", "0612345678", "alice@mail.com", "Paris")
contacts = ajouter_contact(contacts, "Bob", "0755667788",
"bob@mail.com", "Lyon")
afficher_contacts(contacts)

print(rechercher_contact(contacts, "Alice"))
contacts = modifier_contact(contacts, "Alice", "ville", "Marseille")
print(rechercher_par_ville(contacts, "Marseille"))
contacts = supprimer_contact(contacts, "Bob")
afficher_contacts(contacts)
```

2 Exercice 2

Une base de données gère les informations d'une librairie. Elle contient trois tables principales :

- **Auteur**(id_auteur, nom, prenom, pays)
- **Livre**(id_livre, titre, prix, id_auteur)
- **Client**(id_client, nom, ville)

Les relations sont les suivantes :

- un auteur peut avoir écrit plusieurs livres ;
- chaque livre est écrit par un seul auteur ;
- la table `Client` est indépendante (elle ne sera utilisée que pour les requêtes simples).

Le but de ce sujet est de rédiger plusieurs requêtes SQL, certaines portant sur une seule table, et d'autres impliquant une jointure entre deux tables.

1. Afficher le titre et le prix de tous les livres dont le prix est supérieur à **20 euros**, triés du plus cher au moins cher.
2. Afficher les auteurs originaires de **France**. Le résultat doit contenir : le nom, le prénom et le pays.
3. Afficher la liste des livres avec le nom et le prénom de leur auteur. On souhaite les colonnes : `titre`, `nom`, `prenom`.
4. Afficher les titres des livres et les noms de leurs auteurs pour les livres dont le prix est inférieur à **15 euros**.
5. Afficher, pour chaque auteur, le **nombre de livres écrits**. Les colonnes attendues sont : `nom`, `prenom`, `nb_livres`.

   ```
   -- A completer :
   SELECT ...
   FROM Auteur
   JOIN Livre ON ...
   GROUP BY nom, prenom;
   ```

6. Afficher le **prix moyen** des livres présents dans la table `Livre`. La colonne du résultat doit s'appeler `prix_moyen`.