Série N°3 (TP)
(Enregistrements et Fichiers)

Exercice 1 : Gestion des employés

On souhaite créer un programme Python permettant de gérer une liste d'employés. Chaque **employé** est modélisé sous la forme d'un **dictionnaire** contenant les champs suivants :

- **id** : entier (identifiant unique de l'employé)
- **nom** : chaîne de caractères
- **âge** : entier
- **poste** : chaîne de caractères

L'ensemble des employés sera stocké dans une **liste de dictionnaires**. Les données pourront également être sauvegardées et chargées depuis un **fichier texte** nommé "**employes.txt**", où chaque ligne représente un employé, avec les champs séparés par des virgules.

Travail demandé :

1. Définir la structure d'un **employé** à l'aide d'un **dictionnaire**.
2. Ecrire une fonction **saisir_employe( )** qui retourne un dictionnaire représentant un employé saisi par l'utilisateur.
3. Ecrire une fonction **afficher_employe(employe)** qui affiche les informations d'un employé donné.
4. Ecrire le programme principal qui :
   - Demande à l'utilisateur de saisir les informations de **n** employés (n ≤ 10).
   - Stocke ces employés dans une liste.
   - Affiche les **noms** des employés dont **l'âge est strictement supérieur à 30 ans**.
5. Ecrire une fonction **enregistrer_employes(liste, nom_fichier)** qui enregistre la liste des employés dans un fichier texte ("**employes.txt**"), chaque employé étant sur une ligne au format : **id**, **nom**, **age**, **poste**.
6. Ecrire une fonction **charger_employes(nom_fichier)** qui lit les données du fichier texte et retourne la liste des employés (sous forme de dictionnaires).
7. Ecrire une fonction **rechercher_employe(liste, id)** qui recherche un employé par id dans la liste et affiche ses informations s'il existe.
8. Ecrire une fonction **mettre_a_jour_poste(liste, id, nouveau_poste)** qui met à jour le poste d'un employé identifié par son **id**.
9. Ecrire une fonction **supprimer_employe(liste, id)** qui supprime l'employé ayant l'id donné de la liste, puis met à jour le fichier "**employes.txt**".
10. Ecrire une fonction **trier_employes_par_age(liste)** qui trie la liste des employés par âge croissant et affiche la liste triée.

Exercice 2 : Gestion des étudiants

On souhaite développer une application Python pour gérer une liste d'étudiants. Chaque **étudiant** est représenté par un dictionnaire comportant les champs suivants :

- **code** : entier (identifiant unique de l'étudiant)
- **nom** : chaîne de caractères
- **prenom** : chaîne de caractères
- **genre** : caractère ('M' pour masculin ou 'F' pour féminin)
- **moyenne** : réel (note moyenne sur 20)

Les **étudiants** sont stockés dans une **liste de dictionnaires** et peuvent être enregistrés ou chargés depuis un **fichier JSON** nommé "**etudiants.json**".

Travail demandé :

1. Déclarer l'enregistrement **Etudiant** à l'aide d'un **dictionnaire**.
2. Ecrire une fonction **saisir_etudiant( )** qui permet de remplir les champs d'un étudiant saisis par l'utilisateur, et retourne un dictionnaire.
3. Ecrire une fonction **afficher_etudiant(etudiant)** qui affiche les informations d'un étudiant donné.
4. Ecrire un programme principal qui :
   - Demande à l'utilisateur de **saisir** les informations de **n** étudiants (avec n ≤ 100), et les stocke dans une **liste**.
   - Affiche les **noms** des étudiants ayant une moyenne **supérieure ou égale à 10**.
   - Affiche les **informations** de l'étudiant ayant **la moyenne maximale**.
5. Ecrire une fonction **sauvegarder_etudiants(liste, nom_fichier)** qui enregistre la liste des étudiants dans un fichier **JSON** nommé "**etudiants.json**".
6. Ecrire une fonction **charger_etudiants(nom_fichier)** qui lit les données depuis le fichier JSON et reconstitue la liste des étudiants sous forme de dictionnaires.
7. Ecrire une fonction **rechercher_etudiant_par_code(code, nom_fichier)** qui recherche un étudiant dans le fichier **JSON** à partir de son code, et retourne ses informations s'il est trouvé.
8. Ecrire une fonction **mettre_a_jour_moyenne(code, nouvelle_moyenne, nom_fichier)** qui met à jour la moyenne d'un étudiant recherché par son code, puis sauvegarde les données dans le fichier JSON.
9. Ecrire une fonction **supprimer_etudiant(code, nom_fichier)** qui supprime un étudiant identifié par son code, puis sauvegarde le reste des étudiants dans un nouveau fichier JSON nommé "**etudiants_maj.json**".

Exercice 3 : Manipulation des nombres complexes

Un **nombre complexe Z** est entièrement défini par sa partie **réelle** a et sa partie **imaginaire** b, et s'écrit sous la forme : Z = a + bi

Dans cet exercice, les nombres complexes seront représentés à l'aide de **dictionnaires** en Python. Ils seront stockés dans une **liste**, ce qui permettra d'effectuer diverses opérations mathématiques et manipulations. Ces données pourront également être enregistrées ou chargées depuis un **fichier CSV** nommé "**complexes.csv**".

Travail demandé :

1. Déclarer un nombre complexe en utilisant un dictionnaire Python.
2. Ecrire les fonctions suivantes :
   - **partieReel(Z)** : retourne la partie réelle du nombre complexe **Z**.
   - **partieImaginaire(Z)** : retourne la partie imaginaire du nombre complexe **Z**.
   - **Module(Z)** : retourne le module du nombre complexe **Z**, défini par :$|Z| = \sqrt{a^2 + b^2}$.
3. Implémenter les fonctions arithmétiques suivantes :
   - **addition(Z1, Z2) :** retourne la somme de **Z1** et **Z2**.
   - **soustraction(Z1, Z2)** : retourne la différence **Z1-Z2**.
   - **multiplication(Z1, Z2)** : retourne le produit de **Z1** par **Z2**, selon la formule :
   $$(a + bi)(c + di) = (ac - bd) + (ad + bc)i$$
4. Ecrire une fonction **conjuge(Z)** qui retourne le conjugué d'un nombre complexe **Z**, soit $a–bi$.
5. Ecrire une fonction **inverse(Z)** qui retourne l'inverse d'un nombre complexe **Z**, si celui-ci est non nul : $Z^{-1} = \frac{a-bi}{a^2+b^2}$
6. Ecrire une fonction **egalite(Z1,Z2)** qui teste si deux nombres complexes sont égaux (même partie réelle et imaginaire).
7. Ecrire une procédure **afficher(Z)** qui permet d'afficher un nombre complexe sous forme textuelle (ex : 3 + 4i ou 2 - 5i, 7 si imaginaire nul...).

On suppose ensuite que l'on dispose d'un tableau **TC** contenant **N** nombres complexes (N ≤ 100).

8. Afficher le **nombre complexe de plus grand module** dans le tableau TC, puis vérifier si son **conjugué** est également présent dans ce tableau.
9. Calculer :
    - La somme **Zs** de tous les éléments du **TC**.
    - Le produit **Zp** de tous les éléments non nuls du tableau.
10. Calculer la **différence Zs - Zp** et afficher le résultat **uniquement si** cette différence est un **imaginaire pur** (sa partie réelle est nulle).
11. Ecrire une fonction **enregistrer_csv(TC, nom_fichier)** qui enregistre le tableau **TC** dans un fichier **CSV** nommé "**complexes.csv**", chaque ligne contenant deux valeurs : la partie réelle et la partie imaginaire du nombre complexe.
12. Ecrire une fonction **charger_csv(nom_fichier)** qui lit le fichier "**complexes.csv**", et retourne une liste de dictionnaires représentant les nombres complexes lus.
13. Après avoir lu les données depuis le fichier "**complexes.csv**", afficher tous les nombres complexes **dont le module est strictement supérieur à une valeur donnée** (saisie par l'utilisateur).