PAI cours 1

Le but de ce TP est de réviser rapidement les fondamentaux du langage Python. Le cours détaillé est disponible sur https://learnpython.ovh (PDF Langage Python), et **doit** être consulté pendant ce TP.

Consigne fondamentale : documenter chaque fonction avec une docstring.

1. Contact

- Créer un fichier contact_modele.py
- Développer une fonction creer_contact() :
  - *ARGUMENTS* : nom, prénom, adresse
  - *VALEUR DE RETOUR* : dictionnaire avec 3 clés "nom", "prénom", "adresse"
- Tester creer_contact() : ajouter une fonction tester() appelée via le mécanisme suivant :

```
if __name__ == '__main__':
```

Un dictionnaire avec 3 clés "nom", "prénom", "adresse" est appelé **CONTACT** dans la suite de ce TP.

- Développer une fonction enregistrer_contact() :
  - *ARGUMENTS* : objet fichier, **CONTACT**
  - *VALEUR DE RETOUR* : None

Le format d'enregistrement est le suivant :

```
nom///STOP///prénom///STOP///adresse
```

(un contact = une ligne)
(utiliser une variable globale pour stocker le délimiteur, ne pas l'écrire en dur à plusieurs endroits du programme !)
(nota : remplacer "///STOP///" par ";" crée un fichier au format CSV...)

- Tester enregistrer_contact(). Ne pas oublier qu'il faut ouvrir un fichier en écriture au préalable, de préférence via un bloc **with** :

```
with open('carnet.txt', 'w') as f:
```

- Développer une fonction charger_contact():
  - *ARGUMENTS* : objet fichier
  - *VALEUR DE RETOUR* : **CONTACT** (ou None si un contact n'est pas lu, cf ci-

dessous)

Cette fonction lit une ligne (une seule) dans le fichier et renvoie le contact lu.

- Tester charger_contact(). Ne pas oublier que la fonction doit être robuste. Par exemple : détecter la fin de fichier ou une ligne vide, vérifier le format de la ligne lue, etc... Ne pas hésiter à créer à la main des fichiers d'enregistrement corrompus pour tester la robustesse de charger_contact().
- Créer une fonction afficher_contact() pour améliorer l'affichage d'un contact. Utiliser une f-string ou la méthode str.format(). Un contact doit s'afficher sur une ligne.

2. Carnet

Un **CARNET** est une liste de contact. Il est enregistré dans un fichier texte unique (un contact par ligne).

- Développer enregistrer_carnet(), charger_carnet(), afficher_carnet()
- Créer le fichier test_carnet.py. Dans ce fichier, développer un test complet et automatique des fonctionnalités d'un carnet. Un résultat "PASSED" ou "FAILED" doit être affiché.

3. Interface utilisateur

Une interface utilisateur en mode texte est nécessaire pour manipuler facilement un carnet et ses contacts.

- Créer le fichier contact_view.py
- Créer une fonction afficher_menu()
    - *ARGUMENTS* : (pas d'argument)
    - *VALEUR DE RETOUR* : None

Le menu affiché est le suivant :

```
1. Définir le nom de fichier du carnet
2. Charger un carnet
3. Enregistrer le carnet
4. Afficher le carnet
5. Ajouter un contact
6. Supprimer un contact
9. Quitter
```

- Créer une fonction run()
    - *ARGUMENTS* : (pas d'argument)
    - *VALEUR DE RETOUR* : None

Cette fonction affiche le menu, et demande à l'utilisateur de saisir son choix parmi les

options proposées. Après une saisie invalide, le menu et la question sont réaffichés. Après une saisie valide, le traitement associé est exécuté, puis le résultat est affiché avant de réafficher menu et question. Utiliser une boucle **while**.

- Implémenter l'option "Quitter". Pour la saisie, utiliser la fonction input().
- Implémenter l'option "Définir le nom de fichier du carnet". Modifier afficher_menu() avec un argument nom_carnet et son affichage avant celui du menu lui-même (de façon à créer un petit tableau de bord permettant à l'utilisateur d'avoir des informations avant de choisir).
- Implémenter les autres options avec le message "Fonction non implémentée".

Attention ! Chaque option doit être implémentée dans une fonction spécifique et jamais dans run().

4. Fonctions basiques

- Implémenter l'option "Ajouter un contact". Le carnet doit être défini dans run() et fourni comme argument à toutes les fonctions de traitement,y compris celle-ci. Le tableau de bord doit afficher le nombre de contacts du carnet (en plus de son nom).
- Implémenter l'option "Enregistrer le carnet".
- Implémenter l'option "Charger un carnet".
- Implémenter l'option "Supprimer un contact".

5. Fonction de tri

La fonctionnalité de tri est nécessaire. Par défaut, il est impossible de comparer deux dictionnaires. Néanmoins la méthode list.sort() et la fonction native sorted() peuvent être paramétrées pour implémenter le tri souhaité. Pour en savoir plus : lire la documentation de Python.

- Identifier et tester la méthode pour trier une liste de dictionnaires (par exemple dans la fenêtre interactive).
- Ajouter et implémenter l'option "7. Trier le carnet" en utilisant cette méthode (clé de tri : nom du contact).

6. Fonction de recherche

La fonctionnalité de recherche dans le carnet est souhaitable. L'utilisateur doit saisir une chaîne de caractères à rechercher dans nom, prénom ou adresse. L'implémentation de cette fonctionnalité peut être générique :

- Créer une fonction dictionnaire ==> tuple des valeurs de ce dictionnaire.
- Créer une fonction chaîne de caractère, tuple ==> True si la chaîne de caractère est une sous-chaîne d'un élément du tuple, False sinon.
- Ajouter et implémenter l'option "8. Rechercher dans le carnet" en utilisant les deux fonctions élémentaires créées précédemment.

7. Pour les plus rapides

- Remplacer le format d'enregistrement du carnet par un format standard JSON. Utiliser le paquetage standard.
- Modifier le tri pour trier par le prénom puis le nom (actuellement : nom).
- Ajouter le typing dans le modèle pour améliorer la lisibilité et la maintenabilité du logiciel :
    - CONTACT = Dict[str, str]
    - CARNET = List[CONTACT]
    - ...
- Créer une nouvelle version du logiciel utilisant des objets namedtuple en lieu et place des objets dictionnaire. Cf paquetage collections.