TP n°4 *Structures de données imbriquées*

Exercice 1. Echauffement - Fan-club

On donne un dictionnaire `groupePrefere` indiquant pour chaque personne son groupe de musique favori.

```
groupePrefere={'Florent':'Chantal Goya','Celine':'SuperBus',
  'Julien':'ACDC','Denys':'ACDC','Caroline':'Chantal Goya'}
```

1.1. Écrire une fonction `fansDe` qui, à partir d'un tel dictionnaire et d'un nom de groupe, renvoie l'ensemble des fans de ce groupe.

1.2. Écrire une fonction qui à partir d'un dictionnaire `groupePrefere`, renvoie un dictionnaire de fans c'est à dire un dictionnaire dont les clés sont des noms de groupes, et les valeurs sont l'ensemble des fans de chaque groupe.

Exercice 2. Naturalistes en herbe

On a un dictionnaire `regimeAlimentaire` dont les clés représentent des animaux, et les valeurs l'ensemble des choses dont ils peuvent se nourrir. Un autre dictionnaire `nourritureDisponible` indique pour un certain nombre d'endroits, l'ensemble des aliments qu'on y trouve.

```
regimeAlimentaire={
    'Requin':{'Nageur','Sac Plastique','Poisson'},
    'Nageur':{'Poisson','Noisette'},
    'Lion':{'Gazelle'},
    'Ecureuil':{'Noisette'}}
nourritureDisponible={
    'Ocean':{'Poisson','Sac Plastique','Nageur'},
    'Savane':{'Gazelle'},
    'Jardin avec piscine':{'Nageur','Noisette'}
}
```

2.1. Écrire une fonction `inverse(dictionnaire)` qui prend en argument un dictionnaire et inverse les clés et les valeurs. Par exemple, sur un dictionnaire comme `regimeAlimentaire`, `inverse` renvoie un dictionnaire dont les clés sont les aliments et la valeur associée est l'ensemble des animaux qui consomment cet aliment.

2.2. Écrire une fonction `peutSurvivre(regimeAlimentaire,nourritureDisponible)` qui renvoie un dictionnaire dont les clés sont les lieux et les valeurs sont les animaux qui peuvent survivre à cet endroit.

Exercice 3. Rapide et furieux, une série qui se répète un peu

On vous donne le script suivant :

```
#Version 1
def nombre_apparition(chaine, caractere):
    cpt=0
    for char in chaine:
        # ICI
        if caractere == char:
            cpt=cpt+1
    return cpt

def caracteres_en_double(chaine):
    """
    resultat : l'ensemble des caractères qui apparaissent au
    moins 2 fois dans la chaine
    """
    caracteresRepetees = set()
    for caractere in chaine:
        if nombre_apparition(chaine, caractere) > 1:
            caracteresRepetees.add(caractere)
    return caracteresRepetees
```

3.1. Combien de fois passe-t-on par la ligne *# ICI* si on ajoute la ligne suivante ?

```
print(caracteres_en_double("Yolo"))
```

Réponse:

3.2. Combien de fois passe-t-on par la ligne *# ICI* si on ajoute la ligne suivante ?

```
print(caracteres_en_double("Tu lui brises le coeur, je te brise
   la tête !")) # 45 caractères
```

Réponse:

3.3. Sur une chaîne de N caractères, combien de passages fait-on par la ligne *#ICI* ?

Réponse:

Voici deux autres versions de la fonction `caracteres_en_double()`

```python
#Version 2
def caracteres_en_double(chaine):
    """
    resultat : la liste des caractères qui apparaissent au
    moins 2 fois dans la chaine
    """
    dejaVu=[]
    caracteres_repetes=[]
    for caractere in chaine:
        if caractere not in dejaVu:
            dejaVu.append(caractere)
        else:
            caracteres_repetes.append(caractere)
    return caracteres_repetes
```

```python
#Version 3
def caracteres_en_double(chaine):
    """
    resultat : l'ensemble des caractères qui apparaissent au
    moins 2 fois dans la chaine
    """
    dejaVu=set()
    caracteres_repetes=set()
    for caractere in chaine:
        if caractere not in dejaVu:
            dejaVu.add(caractere)
        else:
            caracteres_repetes.add(caractere)
    return caracteres_repetes
```

3.4. Repérer les instructions lentes dans les trois fonctions.

3.5. Laquelle des trois versions est préférable ? Utiliser `timeit` pour comparer le temps d'exécution des trois versions en traçant des courbes.

Exercice 4. Oh les amoureux !

L'Association des Timides Maladifs a décidé de prendre les choses en main ! Chaque membre a écrit sur un petit papier son nom, et le nom de l'être aimé. Ces petits papiers ont été compilés en un dictionnaire dont les clés sont les membres de l'ATM, et la valeur associée à chaque clé est le nom de l'élu·e de son cœur.

```python
ATM={'Armand':'Beatrice','Beatrice':'Cesar','Cesar':'Dalida',
     'Dalida':'Cesar','Etienne':'Beatrice','Firmin':'Henri',
     'Gaston':'Beatrice','Henri':'Firmin'}
```

4.1. Écrire une fonction qui à partir d'un tel dictionnnaire, renvoie les couples dont l'amour est réciproque.

4.2. Un infâme personnage s'est emparé du dictionnaire ! Il va éliminer ses rivaux et rivales ! Et vous allez l'aider... Écrire une fonction `soupirants` qui à partir d'un tel dictionnaire et du nom d'une personne, renvoie toutes les personnes qui sont amoureuses de la personne en question.

Exercice 5. A table ! Implémentez les fonctions de l'exercice 7 du TD 3

Exercice 6. Le retour des timides maladifs

L'Association des Timides Maladifs est de retour. Une branche a fait sécession : il s'agit de l'Association des Timides et Indécis Maladifs. Contrairement aux timides maladifs, les timides indécis maladifs pensent que l'on peut être épris de plusieurs personnes. Ils ont donc décidé de se réunir et d'inscrire chacun sur un petit papier plusieurs noms, ceux de toutes les personnes dont ils sont épris. On a ainsi obtenu un dictionnaire `amoursATIM` qui à chaque personne, associe l'ensemble des objets de son amour.

```
amoursATIM={
  'Armand':{'Beatrice','Dalida'},
  'Beatrice':{'Cesar','Armand'},
  'Cesar':{'Dalida','Gaston'},
  'Dalida':{'Cesar','Armand'},
  'Etienne':{'Beatrice','Firmin'},
  'Firmin':{'Henri','Beatrice','Armand','Dalida'},
  'Gaston':{'Beatrice','Dalida','Cesar'},
  'Henri':{'Firmin','Armand','Cesar','Henri'}}
```

6.1. Écrire à nouveau une fonction `reciproque` qui indique les couples dont l'amour est réciproque

6.2. La société est vraiment trop cruelle. L'idéaliste veut partir sur une île déserte avec uniquement des gens qui s'aiment tous les uns les autres pour y repartir de zéro. Souhaitons-lui bon voyage et donnons-lui un petit coup de main : écrire une fonction `hippies` qui prend en argument un dictionnaire `amours` analogue à `amoursATIM` et une liste de passagers et vérifie qu'ils s'aiment tous les uns les autres.

Exercice 7. Syracuse

On rappelle la définition de la suite de Syracuse (cf TP1) : à partir d'un entier n, c'est la liste d'entiers obtenue ainsi :

- le premier entier est n
- si le dernier entier calculé k est pair, le suivant est sa moitié
- sinon, si cet entier k est impair, le suivant est $3k + 1$
- si le dernier entier calculé est 1, c'est la fin de la liste.

On appelle *temps de vol* d'un entier n le nombre de termes avant d'arriver à 1, si on part de n.

On cherche des nombres «petits» qui aient le plus long temps de vol possible.

7.1. Écrire une fonction `temps_de_vol(n)` qui calcule le temps de vol de l'entier n.

7.2. Écrire une fonction `champion(n)` qui renvoie le nombre inférieur à n qui a le plus long temps de vol.

7.3. Jusqu'à quelle valeur de n l'appel à `champion(n)` prend-il moins d'une seconde ?

Voici une piste pour améliorer le temps de calcul de `champion` : on va garder dans un dictionnaire chaque temps de vol que l'on calcule ; quand on calcule un temps de vol, si on retombe sur un nombre dont on connait le temps de vol, on peut répondre directement sans avoir besoin de calculer les valeurs suivantes de Syracuse.

7.4. Sachant que le temps de vol de 10 est 6, quel est le temps de vol de 20 ? Sachant que le temps de vol de 124 est 108, quel est le temps de vol de 27 ?

7.5. Écrire une fonction `temps_de_vol_avec_precalcul(n, temps_connus)` qui calcule le temps de vol de n grâce aux temps de vols déjà connus, contenus dans le dictionnaire `temps_connus`. Par exemple, pour la question précédente, on ferait un appel comme `temps_de_vol_avec_precalcul (27, {10: 6, 124: 108})`.

7.6. Modifier `champion` pour utiliser `temps_de_vol_avec_precalcul`.