TP n°3 *Ensembles et dictionnnaires*

Récupérer le fichier `TP3.py` sur Célène.

Exercice 1. Performance

On donne le code suivant :

```
def elements_en_commun_v1(liste1, liste2):
    """
    resultat : la liste des éléments communs à 'liste1' et 'liste2'
    """
    en_commun = []
    for element in liste1:
        if element in liste2:
            en_commun.append(element)
    return en_commun
```

```
def elements_en_commun_v2(liste1, liste2):
    """
    resultat : l'ensemble des éléments communs à 'liste1' et 'liste2'
    """
    ensemble1 = set(liste1)
    ensemble2 = set(liste2)
    en_commun=set()
    for element in ensemble1:
        if element in ensemble2:
            en_commun.add(element)
    return en_commun
```

1.1. Repérez les boucles et les instructions *lentes*

On propose le code ci-dessous pour chronométrer le temps d'exécution de ces deux fonctions avec deux listes disjointes de taille 30 (une liste de nombres paires et une liste de nombres impairs) :

```
from timeit import timeit

taille_des_listes =30
liste1 = [2*n for n in range(taille_des_listes)]
liste2 = [2*n+1 for n in range(taille_des_listes)]

temps1 = timeit('elements_en_commun_v1(liste1, liste2)',
    globals=globals(),number=100)*1000
temps2 = timeit('elements_en_commun_v2(liste1, liste2)',
    globals=globals(),number=100)*1000
```

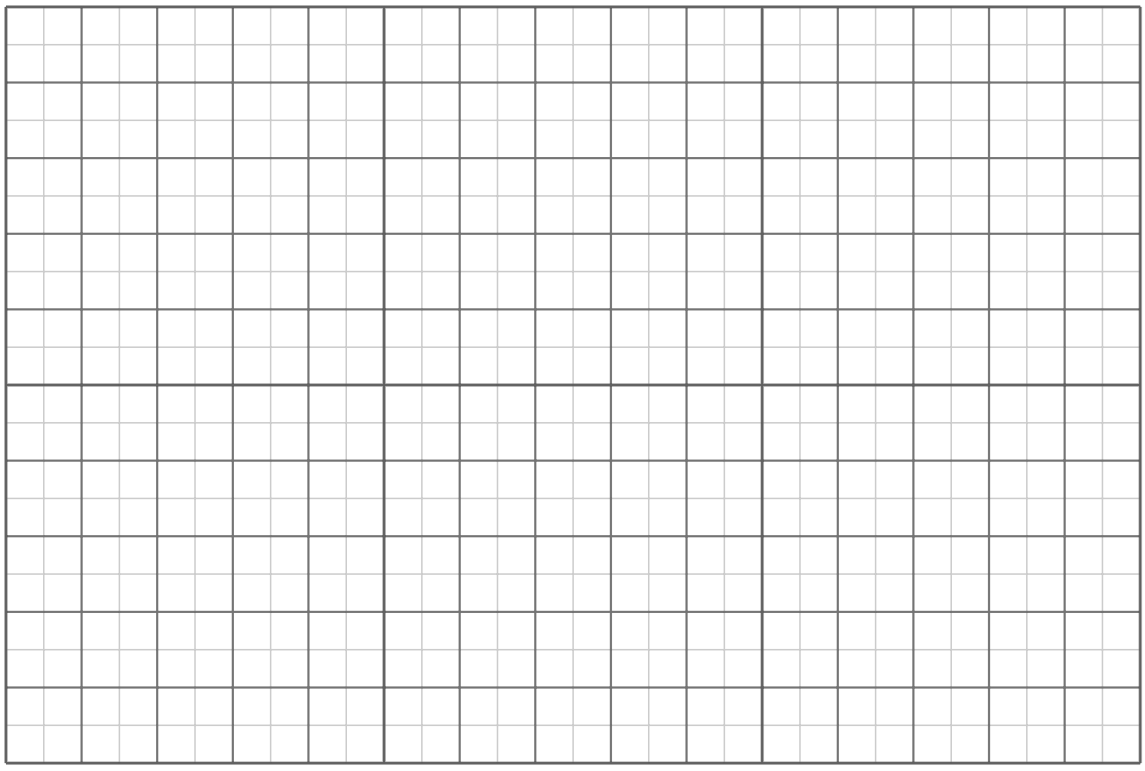
```
print("avec n=",taille_des_listes)
print(temps1)
print(temps2)
```

Attention, le code à chronométrer (le premier argument) est donné *sous forme d'une chaîne de caractères*, et la mention `globals=globals()` est nécessaire pour que vous fonctions puissent être appelées. Avec `number=100`, on indique qu'on veut exécuter ce code 100 fois, puis on multiplie par 1000 pour obtenir un nombre plus lisible.

1.2. Compléter le tableau ci-dessous donnant le temps d'exécution moyen des deux fonctions,

taille_des_listes	10	20	30	40	50	60	70	80	90	100
elements_en_commun_v1										
elements_en_commun_v2										
elements_en_commun_v3										

1.3. Dessiner deux courbes représentant le temps d'exécution des fonctions précédentes en fonction de la taille des données en entrée. Prendre deux listes ou ensembles de même taille. Prendre comme tailles 10, 20, …, 100.



Exercice 2. Fanfare

La fanfare de la Klaipėdos Universitetas recrute ! Trombones, tubas, hautbois et guitarrons sont recherchés. Chacun peut déposer sa candidature pour un ou plusieurs instruments (par ordre de

préférence). Aucun niveau n'est exigé, donc il n'y aura pas d'auditions ! Premier (ou première) arrivé, premier servi.

Nous avons récolté les candidatures, avec pour chaque candidature le nom de la personne qui candidate, et l'instrument.

On peut ainsi avoir les candidatures suivantes :

Candidat·e	Instrument
Kim Gordon	Apito
Superman	Trompette
Frank Zappa	Banjo
Chico Science	Apito
Frank Zappa	Trompette

Nous allons représenter ces candidatures par une liste de couples (nom, instrument).

2.1. Pourquoi ne pas utiliser un dictionnaire ? Pourquoi ne pas utiliser un ensemble de couples ? Donner la représentation de la liste de candidatures donnée en exemple en python.

On veut une fonction `premier_candidat(candidatures, instrument)` qui prend en argument la liste des candidatures et un instrument , et renvoie la première personne qui a candidaté pour cet instrument.

2.2. Compléter le tableau d'exemples suivant :

instrument	`premier_candidat(candidatures, instrument)`
Kazoo	
Apito	
Trompette	
Guitarron	

2.3. Ecrire la fonction `premier_candidat()`

2.4. Écrire une fonction `tous_candidats(candidatures)` qui renvoie l'ensemble de toutes les personnes qui ont candidaté pour faire partie de la fanfare. N'oubliez pas d'écrire au moins un test.

Nous allons maintenant composer la fanfare. Pour cela, nous prenons pour chaque instrument la personne qui a candidaté en premier, sauf si elle joue déjà d'un autre instrument dans la fanfare.

2.5. Dans l'exemple de candidatures qui vous est donné, donner la composition que l'on obtiendra pour la fanfare.

Réponse:

2.6. Écrire une fonction `compose_fanfare(candidatures)` qui, à partir de la liste des candidatures, renvoie un dictionnaire dont les clés sont les personnes qui seront invitées à faire partie de la fanfare, et les valeurs sont les instruments dont ils joueront. N'oubliez pas d'écrire au moins un test.

On va se doter d'une fonction `candidats_malheureux(candidatures)` qui renvoie l'ensemble des candidats qui ne seront pas retenus car il y a un autre candidat pour le même instrument à une position précédente dans le répertoire pour tous les instruments sur lesquels ils ont candidaté.

2.7. Écrire la fonction `candidats_malheureux`.

Indication : on pourra réutiliser `tous_candidats` ainsi que les clés de `compose_fanfare` par une opération ensembliste judicieuse.

2.8. Vérifiez que toutes vos fonctions sont bien testées.

2.9. Repérer dans votre script toutes les boucles et toutes les opérations *lentes*

Exercice 3. Cuisine

On représente une recette de cuisine par un dictionnaire dont les clés sont des noms d'ingrédients, et les valeurs sont la quantité nécessaire de chaque ingrédient.

3.1. Donner un exemple de dictionnaire représentant la recette des crêpes.

On représente un magasin par un dictionnaire dont les clés sont les ingrédients disponibles, et les valeurs le prix d'une quantité unitaire de l'ingrédient.

3.2. Écrire une fonction `recette_possible` qui prend en argument une recette et un magasin et qui indique si on peut s'y procurer tous les ingrédients de la recette.

3.3. Écrire une fonction `prix_recette` qui prend en argument une recette et un magasin et qui donne le prix total pour acheter les ingrédients de la recette dans ce magasin.

Exercice 4. La fanfare fait un rappel (défi)

Après quelques prestations remarquées, la fanfare s'aperçoit d'un problème : il faudrait beaucoup plus de joueurs de triangle pour qu'on puisse les entendre, tandis qu'un seul hélicon suffit à faire *pon pon pon pon*. On décide de refaire le recrutement de la fanfare, en indiquant cette fois de combien de musiciens on a besoin pour chaque instrument.

Voici un exemple des besoins de la fanfare :

Instrument	Nombre
Hélicon	1
Triangle	4
Clarinette	2

Nous représenterons ces besoins par un dictionnaire dont les clés sont les instruments, et les valeurs le nombre d'instrumentistes à recruter.

4.1. Puisque nous allons avoir besoin de plusieurs instrumentistes par instrument, quelle structure de donnée allons-nous utiliser pour représenter la composition de la fanfare ?

On propose de procéder comme suit pour les recrutements : on reçoit les candidatures une à une, et si il y a un besoin pour l'instrument en question, on recrute la personne. Bien sûr, une personne peut candidater plusieurs fois, mais si elle est recrutée pour un instrument, elle ne sera pas recrutée à nouveau pour un autre.

4.2. Écrire une fonction `recrute_besoins(candidatures, besoins)` qui fait le recrutement. Cette fonction renverra un dictionnaire représentant la fanfare après les recrutements.

Exercice 5. Pea pattern sequence

On considère ici une variante de la suite A005150 de l'exercice 7 du TD2. Le principe est que pour passer d'une ligne à la suivante, on compte *toutes* les occurences de chaque chiffre dans la ligne dont on part.

Par exemple, en partant de `l1 = [1, 2, 1, 1, 2, 2, 1, 3, 3]`, on obtient `l2 = [4 1 3 2 2 3]` qui se lit «quatre "1", trois "2", deux "3"».

5.1. Donner le dictionnaire des fréquences de la liste `l1`.

`l2` s'obtient à partir du dictionnaire des fréquences de `l1`, en listant les clés dans l'ordre. Pour parcourir les clés d'un dictionnaire `mon_dico` dans l'ordre, on peut utiliser la boucle suivante :

```
for cle in sorted(mon_dico):
  valeur = mon_dico[cle]
  # traitement de cle et valeur
```

5.2. Écrire une fonction `lit_dico_frequences(d)` qui, à partir du dictionnaire des fréquences, construit une liste sur le modèle de `l2`.

Si l'on répète ce traitement sur une liste de départ, on obtient une suite de listes sur le modèle suivant :

```
[1]
[1 1]
[2 1]
[1 1 1 2]
[3 1 1 2]
[2 1 1 2 1 3]
[3 1 2 2 1 3]
```

5.3. Écrire une fonction `n_lignes(depart, n)` qui donne les n premières lignes en partant de la liste `depart`.

5.4. En partant de la liste `[1]`, quelle est la première ligne qui revient plusieurs fois ?