Séance 8 : Ensembles, dictionnaires et matrices

L1 – Université Côte d'Azur

Exercice 1 — L'ensemble des lettres minuscules (⋆)

1. Définissez l'ensemble `minuscules` contenant les 26 lettres minuscules de l'alphabet.
2. Définissez la fonction `nb_occurrences_minuscules(s)` qui prend en argument une chaîne de caractères `s` et qui renvoie le nombre d'occurences d'une lettre minuscule dans `s`, en testant l'appartenance de chaque caractère de `s` à l'ensemble `minuscules`. Par exemple, `nb_occurrences_minuscules('AaAaBz')` renvoie 3.
3. Définissez la fonction `ensemble_minuscules(s)` qui renvoie l'ensemble des lettres minuscules apparaissant dans `s`. Par exemple, `ensemble_minuscules('AaAaBz')` renvoie `{'a','z'}`.
4. Déduisez-en la fonction `nb_minuscules(s)` qui renvoie le nombre de lettres minuscules apparaissant dans `s`.

Exercice 2 — Liste de contacts (⋆)

On dispose d'une variable `contacts` contenant une liste de contacts d'un smartphone. Cette liste de contacts est stockée à l'aide d'un dictionnaire Python. Par exemple, on pourrait avoir :

```
>>> print(contacts)
{'Chloé': '0601020304', 'Quentin': '0710203040', 'Lyes': '0623344556'}
```

1. Qu'affiche `print(contacts.keys())` ?
2. Qu'affiche `print(contacts.values())` ?
3. Quelle instruction permet de remplacer le numéro de Chloé par `'0611223344'` ?
4. Quelle instruction permet d'ajouter Sarah dans la liste de contacts, dont le numéro est `'0145444342'` ?
5. Quelle instruction permet d'afficher le numéro de Lyes ?
6. Quelle instruction permet d'effacer Chloé du répertoire ?

Exercice 3 — Doubler les valeurs d'une matrice (⋆)

On représente une matrice par la liste de ses lignes (qui sont elles-mêmes représentées par des listes).

Par exemple, `M = [ [1,2,3], [4,5,6] ]` représente la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

On rappelle que vous pouvez utiliser les deux fonctions vues en cours :

- `dimensions(M)` qui renvoie un couple `(n,m)` correspondant au nombre de lignes et de colonnes.
- `matrice_nulle(n,m)` qui renvoie une matrice de dimensions $n \times m$ et ne contenant que des zéros.

1. Quelle instruction permet de remplacer le 3 par un 0 ?

```
M[0][2] = 0
```

2. Si `M` est une matrice définie dans Python, que donne le calcul `2 * M` dans la console Python ?
   *M1 et M2 sont vues comme des listes par Python, donc M1 + M2 est la concaténation des 2 listes. De plus, par définition, 2*M = M + M. On obtient donc :*

```
>>> M+M
[[1, 2, 3], [4, 5, 6], [1, 2, 3], [4, 5, 6]]
>>> 2*M
[[1, 2, 3], [4, 5, 6], [1, 2, 3], [4, 5, 6]]
```

   On obtient la matrice suivante : $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

3. Écrivez une fonction `doubler(M)` qui *modifie* la matrice `M` et double chacun de ses coefficients.

```
# Cette fonction est dans le cours
def dimensions(M):
    return (len(M),len(M[0]))

def doubler(M):
    (n, m) = dimensions(M)
    for i in range(n):
        for j in range(m):
            M[i][j] = 2 * M[i][j]
```

4. Écrivez une fonction `double(M)` qui *renvoie* une nouvelle matrice correspondant à la matrice `M` dans laquelle les coefficients ont été doublés.

```
# Cette fonction aussi est dans le cours !
def matrice_nulle(n,m):
    A=[]
    for ligne in range(n):
        L=[]
        for colonne in range(m):
            L.append(0)
        A.append(L)

def double(M):
    (n, m) = dimensions(M)
    D = matrice_nulle(n,m)
    for i in range(n):
        for j in range(m):
            D[i][j] = 2 * M[i][j]
    return D
```

Exercice 4 — Carré magique (⋆⋆)

Un carré magique est une matrice $n \times n$ d'entiers telle que la somme de chaque ligne, de chaque colonne, et des deux diagonales est une constante S. Par exemple, pour $n = 3$, on a le carré magique ci-contre de somme constante $S = 15$.

2	7	6
9	5	1
4	3	8

On représente une matrice par une liste de listes. Par exemple, le carré magique ci-contre correspond à `cm = [ [2,7,6] , [9,5,1] , [4,3,8] ]`

1. Écrivez une fonction `est_carré(M)` qui prend en argument une liste de listes `M` et qui renvoie `True` si `M` est une matrice carrée et `False` sinon.

```python
def est_carré(M):
    if M==[]:
        return False
    n = len(M)
    for ligne in M:
        if len(ligne) != n:
            return False
    return True
```

2. Écrivez une fonction `est_magique(M)` qui prend en argument une liste de listes `M` et qui renvoie `True` si `M` est un carré magique et `False` sinon.

   Pour augmenter la lisibilité du code, on écrit des fonctions permettant d'accéder aux lignes, au colonnes et aux diagonales.

```python
def ligne(M,i):
    return M[i]

def colonne(M, j):
    (n,m) = dimensions(M)
    return [ M[i][j] for i in range(m)]

# Renvoie la diagonale de la matrice M. k=0 ou 1 suivant si on veut la
# diagonale de gauche à droite ou de droite à gauche

def diagonale(M,k):
    (n,m) = dimensions(M)
    if k==0:
        return [ M[i][i] for i in range(n)]
    else:
        return [ M[n-1-i][i] for i in range(n)]
```

```python
def est_magique(M):
    if not est_carré(M):
        return False
    (n,m) = dimensions(M) # Forcément n=m
    S = sum(ligne(M,0))
    # vérification des lignes
    for i in range(n):
        if sum(ligne(M,i)) != S:
            return False
    # vérification des colonnes
    for j in range(n):
        if sum(colonne(M,j)) != S:
            return False
    # vérification des diagonales
    for k in range(2): # k=0 ou 1
        if sum(diagonale(M,k)) != S:
            return False
    # Si tous les tests ont été passé avec succés
    return True
```

Un version plus élégante à écrire (mais moins efficace à l'exécution) en utilisant des ensembles. Remarque : $A \cup B$ se note en python $A|B$.

```python
def est_magique(M):
    if not est_carré(M):
        return False
    (n,m) = dimensions(M) # Forcément n=m
    El = { sum(ligne(M,i)) for i in range(n) }
    Ec = { sum(colonne(M,j)) for j in range(m) }
    Ed = { sum(diagonale(M,k)) for k in range(2) }
    return len( El | Ec | Ed ) == 1
```

Exercice 5 — Liste de contacts inversée (⋆)

1. Écrivez une fonction `inverse_liste_contacts(contacts)` qui prend en argument une liste de contacts comme dans l'exercice précédent et qui renvoie la liste indexée par les numéros au lieu des noms. Par exemple :

```
>>> inverse_liste_contacts(contacts)
{'0601020304': 'Chloé', '0710203040': 'Quentin', '0623344556': 'Lyes'}
```

```python
def inverse_liste_contacts(contacts) :
    res = dict()
    for nom in contacts :
        tél = contacts[nom] # ici nom est la clé et tel la valeur
        res[tél] = nom      # ici tél est la clé et nom la valeur
    return res
```

2. Écrivez une fonction `affiche_liste_appels(L ,contacts)` qui prend en arguments une liste de couples de chaînes de caractères de la forme `(date,numero)` et une liste de contacts, et qui affiche la liste des appels reçus en indiquant si possible le nom de la personne qui a appellé.

   Par exemple, on aura en prenant `L = [ ('10:03', '0623344556'), ('9:45', '0623344556'), ('hier', '0800123123'), ('20/11', '0623344556')]` :

```
>>> affiche_liste_appels(L , contacts2)
10:03 Quentin
9:45 Quentin
hier 0800123123
20/11 Chloé
```

```
def affiche_liste_appels(L , contacts) :
    contacts_inv = inverse_liste_contacts2(contacts)
    for e in L :
        (heure,tel)=e
        if tel in contacts_inv :
            nom = contacts_inv[tel]
            print(heure, nom)
        else :
            print(heure ,tel)
```

Exercice 6 — Liste d'associations et mémoïsation (⋆⋆)

On considère maintenant une autre représentation de la liste de contacts : une liste d'association. Ainsi, la liste de contacts précédente correspond à une variable `lst_contacts` qui vaudra par exemple

```
lst_contacts= [('Chloé','0601020304'), ('Quentin','0710203040'), ('Lyes','0623344556')]
```

1. Écrivez une fonction `trouve_numéro(lst_contacts,nom)` qui prend en argument une liste de contact représentée par une liste d'association et un nom, et qui renvoie le numéro de téléphone correspondant, ou -1 si le nom n'est pas dans la liste.

```
def trouve_numéro(contacts,nom) :
    for e in contacts :
        (nom2,tel) = e
        if nom == nom2 :
            return tel
    return -1
```

2. Écrivez une fonction `trouve_numéro_mémo(lst_contacts,nom)` qui fait la même chose mais qui mémoïse les résultats des recherches précédentes. Vous introduirez une variable globale `contacts` qui contiendra, sous la forme d'un dictionnaire, la liste des contacts recherchés par les appels précédents de `trouve_numero_memo`.

```
memo = dict()

def trouve_numéro_mémo(contacts,nom) :
    if not nom in memo : # Si ce n'est pas dans le cache, on le cherche et on l'ajoute
        for e in contacts :
            (nom2,tel) = e
            if nom == nom2 :
                memo[nom] = tel
                break # on a trouvé, inutile de continuer la recherche
    return memo[nom]
```

3. Quel est l'intérêt de cette fonction mémoïsée ?
   intérêt : calculer `contacts[nom]` est en temps « quasi constant » tandis que la recherche par parcours est moins efficace (en temps linéaire).

Exercice 7 — Fibonacci mémoïsé (⋆⋆)

On considère la fonction suivante calculant le n-ième terme de la suite de Fibonacci.

```python
def fibo(n) :
    print(f"début calcul de fibo({n})")
    if n < 2 :
        res = 1
    else :
        res =  fibo(n-1) + fibo(n-2)
    print(f"fin calcul de fibo({n})")
    return res
```

1. Qu'affiche `fibo(5)` ?

```
>>> fibo(5)
début calcul de fibo(5)
début calcul de fibo(4)
début calcul de fibo(3)
début calcul de fibo(2)
début calcul de fibo(1)
fin calcul de fibo(1)
début calcul de fibo(0)
fin calcul de fibo(0)
fin calcul de fibo(2)
début calcul de fibo(1)
fin calcul de fibo(1)
fin calcul de fibo(3)
début calcul de fibo(2)
début calcul de fibo(1)
fin calcul de fibo(1)
début calcul de fibo(0)
fin calcul de fibo(0)
fin calcul de fibo(2)
fin calcul de fibo(4)
début calcul de fibo(3)
début calcul de fibo(2)
début calcul de fibo(1)
fin calcul de fibo(1)
début calcul de fibo(0)
fin calcul de fibo(0)
fin calcul de fibo(2)
début calcul de fibo(1)
fin calcul de fibo(1)
fin calcul de fibo(3)
fin calcul de fibo(5)
8
```

2. Écrivez une fonction `fibo_memo()` qui mémoïse les résultats des appels antérieurs dans un dictionaire. Faites afficher les débuts et fin d'appels comme ci-dessus. Qu'affiche `fibo_memo(5)` ?

```python
mem = dict()
def fib_memo(n):
    print('début calcul de fibo_memo({})'.format(n))
    if n==0 or n==1:
        mem[n] = 1
    elif n not in mem:
        mem[n] = fib_memo(n-1)+fib_memo(n-2)
    print('fin calcul de fibo_memo({})'.format(n))
    return mem[n]

print('Exécution de fibo_memo(5)')
fib_memo(5)
```

```
>>> fib_memo(5)
début calcul de fibo_memo(5)
début calcul de fibo_memo(4)
début calcul de fibo_memo(3)
début calcul de fibo_memo(2)
début calcul de fibo_memo(1)
fin calcul de fibo_memo(1)
début calcul de fibo_memo(0)
fin calcul de fibo_memo(0)
fin calcul de fibo_memo(2)
début calcul de fibo_memo(1)
fin calcul de fibo_memo(1)
fin calcul de fibo_memo(3)
début calcul de fibo_memo(2)
fin calcul de fibo_memo(2)
fin calcul de fibo_memo(4)
début calcul de fibo_memo(3)
fin calcul de fibo_memo(3)
fin calcul de fibo_memo(5)
8
```

3. *Chez vous, avec votre machine.* Commentez les `print` dans les deux fonctions, puis comparez les temps de calcul de `fibo(30)` et `fibo_memo(30)`.

```python
from time import time

def chronomètre(f,n) :
    t1 = time()
    f(n)
    t2 = time()
    return t2-t1


mem = dict()
def fibo_memo(n):
    if n==0 or n==1:
        mem[n] = 1
    elif n not in mem:
        mem[n] = fibo_memo(n-1)+fibo_memo(n-2)
    return mem[n]

def fibo(n) :
    if n < 2 :
        res = 1
    else :
        res = fibo(n-1) + fibo(n-2)
    return res
```

```
>>> d1 = chronomètre(fibo,30)
>>> d2 = chronomètre(fibo_memo,30)
>>> d1/d2
13593.5
```

Exercice 8 — Liste de contacts inversée avec répétitions (⋆⋆ — ⋆ ⋆ ⋆)

On revient sur la fonction `inverse_liste_contacts` de l'exercice ci-dessus.

1. Que renvoie votre fonction pour la liste de contacts suivante ?

```python
contacts2 = {'Maison Chloé': '0901020304', 'Maison Lyes': '0901020304',
             'Alex': '0412345678'}
```

```
>>> inverse_liste_contacts(contacts2) # On a perdu Chloé !
{'0901020304': 'Maison Lyes', '0412345678': 'Alex'}
```

2. Écrivez la fonction `inverse_liste_contacts2(contacts)` qui prend en argument une liste de contacts comme dans l'exercice précédent et qui renvoie la liste indexée par les numéros au lieu des noms en listant les noms de toutes les personnes à qui appartient le numéro. Par exemple, `inverse_liste_contacts2(contacts2)` renverra

```
>>> inverse_liste_contacts2(contacts2)
{'0901020304': ['Maison Chloé', 'Maison Lyes'], '0412345678': ['Alex']}
```

```python
def inverse_liste_contacts2(contacts) :
    res = dict()
    for nom in contacts :
        tél = contacts[nom]
        if res.get(tél,None)==None:
            res[tél] = [nom]
        else:
            res[tél].append(nom)
    return res
```

Exercice 9 — Le jeu de Nim (⋆ ⋆ ⋆)

Le jeu de Nim que l'on considère fait s'affronter deux joueurs qui doivent à tour de rôle retirer des allumettes d'un tas contenant initialement n allumettes. À achaque tour, un joueur peut retirer 1, 2 ou 3 allumettes. Le joueur qui retire la dernière allumette a perdu.

Par exemple, en partant de 5 allumettes, on a la partie 5,2,1,0 perdue par le joueur qui commence. Dans ce cas précis, le joueur qui joue en second a même une stratégie gagnante : quel que soit le nombre d'allumettes que le joueur qui commence retire au premier tour, il peut se débrouiller pour laisser une seule allumette à la fin du second tour.

1. Écrivez une fonction `nim_gagnant(n)` qui renvoie `True` si le joueur qui commence a une stratégie gagnante et `False` sinon. Par exemple, `nim_gagnant(5)` renvoie `False` mais `nim_gagnant(4)` renvoie `True`.

```
def nim_gagnant(n):
    if n<=0:
        return True
    for i in [1,2,3]:
        if not nim_gagnant(n-i):
            return True

    return False
```

2. Écrivez une fonction `nim_memo(n)` qui renvoie les mêmes résultats même en utilisant la mémoïsation.

```
mem=dict()

def nim_memo(n):
    if n<=0:
        return True
    if n in mem:
        return mem[n]
    for i in [1,2,3]:
        if not nim_memo(n-i):
            mem[n]=True
            return True
    mem[n]=False
    return False
```

3. Comparez le temps de calcul pour `n=33`.

```
>>> d1 = chronomètre(nim_gagnant,33)
>>> d2 = chronomètre(nim_memo,33)
>>> d1/d2 # La différence est impressionnante (un facteur d'environ 200 000)
140563.6923076923
```