

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’occurrences 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 :

```
1 >>> print(contacts)
2 {'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 ?
2. Si `M` est une matrice définie dans Python, que donne le calcul `2 * M` dans la console Python ?
3. Écrivez une fonction `doubler(M)` qui *modifie* la matrice `M` et double chacun de ses coefficients.
4. Écrivez une fonction `double(M)` qui *renvoie* une nouvelle matrice correspondant à la matrice `M` dans laquelle les coefficients ont été doublés.

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.
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.

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 :

```
1 >>> inverse_liste_contacts(contacts)
2 {'0601020304': 'Chloé', '0710203040': 'Quentin', '0623344556': 'Lyes'}
```

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 appelé.

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

```
1 >>> affiche_liste_appels(L , contacts2)
2 10:03 Quentin
3 9:45 Quentin
4 hier 0800123123
5 20/11 Chloé
```

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.
2. Écrivez une fonction `trouve_numéro_mémo(lst_contacts, nom)` qui fait la même chose mais qui mémorise 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`.
3. Quel est l’intérêt de cette fonction mémoïsée ?

Exercice 7 – Fibonacci mémoisé (**)

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

```
1 def fibo(n) :
2     print(f"début calcul de fibo({n})")
3     if n < 2 :
4         res = 1
5     else :
6         res = fibo(n-1) + fibo(n-2)
7     print(f"fin calcul de fibo({n})")
8     return res
```

1. Qu’affiche `fibo(5)` ?
2. Écrivez une fonction `fibo_memo()` qui mémoise les résultats des appels antérieurs dans un dictionnaire. Faites afficher les débuts et fin d’appels comme ci-dessus. Qu’affiche `fibo_memo(5)` ?
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)`.

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 ?

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

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

```
1 >>> inverse_liste_contacts2(contacts2)
2 {'0901020304': ['Maison Chloé', 'Maison Lyes'], '0412345678': ['Alex']}
```

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 chaque 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`.
2. Écrivez une fonction `nim_memo(n)` qui renvoie les mêmes résultats même en utilisant la mémoisation.
3. Comparez le temps de calcul pour $n=33$.