Séance 8: Ensembles, dictionnaires, matrices

L1 – Université Nice Sophia Antipolis

Objectifs:

- Utilisation simple d'un ensemble Python
- Utilisation simple d'un dictionnaire Python
- Introduction aux matrices
- Mémoïsation
- Programmation dynamique

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* `alphabet`*. 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 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,* `print(contacts)` *affiche quelque chose comme*

```
{'Chloé': '0601020304' , 'Quentin': '0710203040' , 'Lyes' : '0623344556
    ' , 'Alex': '0412345678'}
```

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* $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

1. *Quelle instruction permet de remplacer le 3 par un 0 ?*
2. *Écrivez une fonction* `dimensions(M)` *qui renvoie le couple constitué du nombre de lignes et du nombre de colonnes de* `M`.

```
>>> dimensions([ [1,2,3], [4,5,6] ])
(2,3)
```

3. *Si* `M` *est une matrice définie dans Python, que donne le calcul* `2 * M` *dans la console Python ?*
4. *Écrivez une fonction* `doubler(M)` *qui modifie la matrice* `M` *et double chacun de ses coefficients.*

□

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 $S = 15$.
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] ]`

2	7	6
9	5	1
4	3	8

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

□

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)` *renvoie quelque chose comme*

```
{'0601020304': 'Chloé' , '0710203040': 'Quentin' , '0623344556': '
    Lyes' , '0412345678': 'Alex' }
```

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*

```
L = [ ('10:03' , '0412345678') , ('9:45' , '0412345678') , \
      ('hier', '0800123123') , ('20/11' , '0623344556') ]
affiche_liste_appels(L , contacts2)
```

affiche

```
10:03 Alex
9:45 Alex
hier 0800123123
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*

```
[ ('Chloé' , '0601020304') , ('Quentin' , '0710203040') , \
  ('Lyes' , '0623344556') , ('Alex': '0412345678') ]
```

1. *Écrivez une fonction* `trouve_numero(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_numero_memo(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`*.*
3. *Quel est l'intérêt de cette fonction mémoïsée ?*

□

Exercice 7 (Fibonacci mémoïsé, ⋆⋆)

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

```
def fibo(n) :
    print('début calcul de fibo(' , n , ')')
    if n < 2 :
        res = 1
    else :
        res =  fibo(n-1) + fibo(n-2)
    print('fin calcul de fibo(' , n , ')')
    return res
```

1. *Qu'affiche* `fibo(5)` *?*
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)` *?*
3. *Si vous avez une machine, commentez les* `print` *dans les deux fonctions, puis comparez les temps de calcul de* `fibo(30)` *et* `fibo_memo(30)`*.*

□

Exercice 8 (Le jeu de Nim, ⋆⋆⋆)

Soit une liste L d'entiers strictements positifs contenant 1 ; dans l'exemple ci-dessous, on prendra `L=[1,2,3]`*. 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. A chaque tour, un joueur peut retirer un nombre k d'allumettes, où k est un entier de L. 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.*

Écrivez une fonction `nim_gagnant(L,n)` *qui renvoie* `True` *si le joueur qui commence a une stratégie gagnante. Par exemple,* `nim_gagnant([1,2,3],5)` *renvoie* `False` *mais* `nim_gagnant([1,2,3],4)` *renvoie* `True`*. Votre fonction devra donner une réponse en temps raisonnable (linéaire en* `n * len(L)`*).*

□