? À rendre LUNDI 5 janvier 2026

Devoir Maison n°5

Partie I Travail à faire

■ Si vous êtes en difficulté (soit environ moins de 8 au DS et/ou si vous vous sentez en difficulté en info) :

- Prévoir de travailler l'info trois fois dans les vacances, à 3-4 jours d'écart ;
- travailler sérieusement la fiche « **Les indispensables** » : **parties II Listes ; III Les dictionnaires** ;
  - essayer d'écrire les fonctions demandées ;
  - vérifier sur le corrigé en ligne ;
  - refaire quelques jours plus tard ;
  - jusqu'à y arriver parfaitement.
- Reprendre l'**exercice 1 du DS**, le refaire en s'aidant de la fiche « les indispensables » et du cours de programmation dynamique.
- Reprendre les questions suivantes de **SQL du DS**, en s'appuyant sur le cours et les commentaires écrits sur votre copie : **Q12, Q13, Q15**.
- Traiter la **partie II ci-dessous**.
- Traiter dans la **partie III Révisions sur la récursivité, les questions Q1 et Q2**.

■ Sinon :

- Faire les **questions de révision sur le range (partie II) sur la récursivité (partie III) ci-dessous**.
- **Reprendre le DS n°1.**
  - Exercice n°1.
    - Q1 : on demandait le minimum et le rang ;
    - Q6 : combien de cases a le tableau que l'on veut remplir ? quel est le dernier rang en ligne ? en colonne ? quelle valeur indiquer dans le range pour parcourir toutes les lignes ? toutes les colonnes ?
  - Exercice n°2. python Q1, Q2, Q5, Q9, Q11
    - Q1 : c'est une somme !
    - Q2 : reprendre le DM où ce tri a été révisé.
    - Q9 : $S(C,k)$ n'est pas une fonction ! et la fonction demandée s'appelle `sol_rec` qui est récursive, qui doit donc s'appeler elle-même. C'est elle qui renvoie la valeur de $S(C,k)$.

```
def sol_rec(C,k):
    if ...=...: # condition d'arrêt
        return .....
    else :
        vk = ... # valeur du concert k
        a = .... # valeur totale si le concert k fait partie de la solution optimale, avec appel récursif sur les concerts en allant jusqu'au rang du compatible avec le concert k
        b = .... # valeur totale si le concert k  ne fait partie de la solution optimale, appel récursif au rang précédent du concert k
        return ....  # on veut la valeur maximale entre les deux possibilités précédentes
```

  - Exercice n°2. SQL : Q12 à Q17.
    - Q14 : attention, la première ligne est numérotée 0.

— Q15, Q16, Q17 : attention aux noms de colonnes ambiguës lors des jointures. Et attention c'est `nom_table.nom_colonne`.

— Q17 : traduction = « il faut compter le nombre de concerts par festivals » , puis récupérer ceux qui en ont au moins 15. Il faut donc commencer par gr...er, et comme ensuite on impose une condition sur un gr...ement, il faut utiliser H....G.

Partie II Et le range dans tout ça ?

Q1. Je dois parcourir une liste `L` entièrement.

(a) Quel est le premier rang ? Quelle est la valeur minimale de `i` pour `L[i]` ?

(b) Quel est le dernier rang ? Quelle est la valeur maximale de `i` pour `L[i]` ?

(c) Dans la boucle j'accède à `L[i]` uniquement, qu'indiquer dans le `range` ?

Q2. Je dois parcourir une liste `L` entièrement, dans la boucle je dois accéder à `L[i]` et `L[i-1]`.

(a) Pour `L[i]` : quelle est la valeur minimale que peut prendre `i` ? quelle est la valeur maximale que peut prendre `i` ?

(b) Pour `L[i-1]` :

Quelle est la valeur minimale que peut prendre `i-1` ? et donc `i` ?

Quelle est la valeur maximale que peut prendre `i-1` ? et donc `i` ?

(c) Par conséquent, compléter la boucle `for` suivante permettant de parcourir toute la liste sans en sortir : `for i in range(...,...):`

Q3. Je dois parcourir une liste `L` entièrement, dans la boucle je dois accéder à `L[i]` et `L[i+1]`.

(a) Pour `L[i]` : quelle est la valeur minimale que peut prendre `i` ? quelle est la valeur maximale que peut prendre `i` ?

(b) Pour `L[i+1]` :

Quelle est la valeur minimale que peut prendre `i+1` ? et donc `i` ?

Quelle est la valeur maximale que peut prendre `i+1` ? et donc `i` ?

(c) Par conséquent, compléter la boucle `for` suivante permettant de parcourir toute la liste sans en sortir : `for i in range(...,...):`

Partie III Révisions sur la récursivité

À retenir

Une fonction récursive est une fonction qui s'appelle elle-même.

```
def f_rec(a,b,c):
    if a==25 : # condition d'arrêt qui porte sur a ou b ou c
        return 0 # valeur pour a=25
    else :
        return f(a+1,b,c) # appel récursif de f_rec sur un rang
   précédent de a ou b ou c
```

Q1. Écrire la fonction récursive `factorielle(n:int)->int` qui renvoie la valeur de $n!$, en exploitant le fait que $n! = n \times (n-1)!$, et $0! = 1$.

```
def factorielle(n):
    if .... :  # condition d'arrêt
        return ....
    else :
        return ..... # appel récursif
```

Comment ça fonctionne ? Les instructions en attente sont empilées, avant d'être dépilées (la structure informatique derrière est une pile).
> `factorielle(3)` demande `3*factorielle(2)`
– – > `factorielle(2)` demande `2*factorielle(1)`
– – – > `factorielle(1)` demande `1*factorielle(0)`
– – – > calcul de 1!
– – > calcul de 2!
> calcul de 3!

Q2. Écrire la fonction récursive `fibo(n:int)->int` qui renvoie la valeur de F_n de la suite de Fibonacci définie par $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$.

```
def fibo(n):
    if .... : #  condition d'arrêt
        return ....
    elif .... : # condition d'arrêt
        return  ....
    else :
        return .....  # appel récursif
```

Q3. Une suite (c_n) plus subtile : $c_0 = 2$ et $c_{n+1} = \begin{cases} \frac{c_n}{2} & \text{si} \quad c_n \text{ est pair} \\ 3c_n + 1 & \text{si} \quad c_n \text{ est impair} \end{cases}$

Que l'on peut réécrire, pour $n \geq 1$: $c_n = \begin{cases} \frac{c_{n-1}}{2} & \text{si} \quad c_{n-1} \text{ est pair} \\ 3c_{n-1} + 1 & \text{si} \quad c_{n-1} \text{ est impair} \end{cases}$

Rappel, `n%2` renvoie le reste de la division euclidienne par 2.

```
def suite(n):
    if .... : #  condition d'arrêt
        return ....
    else :
        prec = .... # valeur de c_(n-1) récupéré via appel récursif
        if ..... :# si pair
            return ....
        else :
            return ....
```

Q4. Un tri récursif : le tri fusion. On suppose avoir une fonction `fusion(L1:list,L2:list)->list` qui à partir des deux listes `L1` et `L2` triées dans le même ordre, renvoi une liste triée dans le même ordre.

Écrire une fonction récursive `tri_fusion(L:list)->list` qui trie par ordre croissant la liste `L` sur le principe : la liste `L` est coupée en deux au milieu, chaque moitié est triée récursivement, puis on fusionne les listes grâce à la fonction `fusion`.

Indice : le milieu d'une liste de longueur est le quotient de la division euclidienne de la longueur de la liste par 2.

```
def tri_fusion(L):
    n = len(L)
    if ..... :  # condition d'arrêt si L a moins d'un élément, elle est
   déjà triée
        return ...
    else :
        L0 = ..... ( L[0:...] ) # tri récursif de la première moitié de L
        L1 =  ..... ( L[...:] ) # tri récursif deuxième moitié de L
        return ..... (L0,L1) # fusion des deux listes triées
```

Ce **tri n'est pas en place** (la liste `L` n'est pas modifiée, une autre est créée), de complexité moyenne $O(n \ln(n))$.

Q5. Un autre tri récursif : le tri bulle. À chaque parcours de la liste, on fait monter la plus grande valeur de la partie qui reste à trier, au bout de la partie qui reste à trier.

```
def tri_bulle(L:list)->list:
    n = len(L)
    if ..... :  # condition d'arrêt si L a moins d'un élément, elle est déjà triée
        return ...
    else : # il faut parcourir la liste jusqu'au dernier rang, pour y placer la plus grande valeur de la liste L
        for i in range(... ,...) : # parcours de L ATTENTION à la valeur maximale de i dans le range vue le contenu de la boucle
            if L[i]>L[i+1]: # l'élément précédent est plus grand que le suivant
                ..... , .... = ..... , ..... # permutation des éléments de rang i et i+1
        # à la fin de la boucle, le plus grand élément de L est arrivé au rang n-1 de L
        return ..... (L[:...]) + [ L[...] ] # appel récursif sur la liste L privée de son dernier élément (qui est à la bonne place), avec concaténation de ce dernier élément.
```

C'est un **tri en place** (=modification de la liste `L`), de **complexité quadratique**.