Définition 1 :
La plus longue sous-chaîne commune (PLSC) de deux chaînes de caractères s_1 et s_2 est une sous-chaîne extraite s de s_1 et s_2 de taille maximum.
On conserve l'ordre des lettres mais s n'est pas nécessairement formées de lettres consécutives des chaînes initiales. Exemple

```
>>> afficher_seq_commune("abcdefghij", "gceggbhaajad")
'ceghj'
```

$s1$ et $s2$ désigneront désormais deux chaînes de caractères de longueurs respectives $n1$ et $n2$.

Méthode récursive naïve

Question 1 : Écrire une fonction **récursive** `plsc_rec(s1 : str, s2 : str) -> str` prenant en entrée deux chaînes $s1$ et $s2$ et retournant la PLSC de `s1` et `s2`.

Question 2 : Notons $n1, n2$ les longueurs des chaînes $s1$ et $s2$.
Déterminer, au meilleur et au pire des cas, la complexité de `plsc_rec`.

Méthode utilsant la programmation dynamique par mémoïsation

Définition 2 :
Résoudre le problème $\mathscr{P}$ par une méthode de programmation dynamique consiste à trouver des solutions optimales à certains sous problèmes de $\mathscr{P}$ pour lesquels il y a souvent des chevauchements puis en déduire (facilement) la solution optimale de $\mathscr{P}$.
On distingue deux approches (qui ne s'opposent pas radicalement) :

- On peut s'appuyer sur une méthode ascendante (**bottom-up**) par tabulation.
  On détermine un ordre de calcul. On chemine (souvent de façon itérative) jusqu'à obtention de la solution du problème d'origine en sauvegardant les résultats dans un tableau.
- On peut s'appuyer sur une méthode descendante (**top-down**) par mémoïsation.
  On part directement du problème cherché. Sa résolution (récursive) entraîne la résolution de sous problèmes dont on mémorise les solutions pour ne pas dupliquer inutilement les calculs (soucis d'efficacité). C'est le principe de la mémoïsation.

Définition 3 :
On appelle matrice des longueurs des plus longues séquences communes de $s1$ et $s2$ la matrice m à $n1$ lignes et $n2$ colonnes telle que $m[i,j]$ soit la longueur de la PLSC de $s1[:i]$ et $s2[:j]$.

Remarque 1 :
- $s1[:i]$ désigne donc le mot formé des i premières lettres de $s1$.
- $s1[:0]$ et $s2[:0]$ sont les mots vides.

Question 3 : Recopier et compléter la relation ci-dessous :

$$\forall (i,j) \in [\![0,n1]\!] \times [\![0,n2]\!] : m[i,j] = \begin{cases} \ldots & \text{si} \quad i=0 \text{ ou } j=0 \\ \ldots & \text{si} \quad i,j>0 \text{ et } s1[i]=s2[j] \\ \ldots & \text{si} \quad i,j>0 \text{ et } s1[i]\neq s2[j] \end{cases}$$

Question 4 : Écrire une fonction `table_seq_commune(s1 : str, s2 : str) -> np.array` prenant en entrée deux chaînes $s1$ et $s2$ et retournant la matrice m des longueurs des plus longues séquences communes de $s1$ et $s2$.

Exemple

```
>>> table_seq_commune("abcdefghij", "gceggbhaajad")
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.],
       [0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2.],
       [0., 0., 1., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [0., 0., 1., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [0., 1., 1., 2., 3., 3., 3., 3., 3., 3., 3., 3.],
       [0., 1., 1., 2., 3., 3., 3., 4., 4., 4., 4., 4.],
       [0., 1., 1., 2., 3., 3., 3., 4., 4., 4., 4., 4.],
       [0., 1., 1., 2., 3., 3., 3., 4., 4., 4., 5., 5.]])
```

Question 5 : Écrire une fonction `longueur_seq_commune(s1 : str, s2 : str) -> int` prenant en entrée deux chaînes $s1$ et $s2$ et retournant la longueur de la plus longue sous-chaîne commune de $s1$ et $s2$.

Question 6 : Déterminer la complexité de la fonction `longueur_seq_commune`.

Pour obtenir la `PLSC` de $s1$ et $s2$, on crée une liste vide `lst` puis on effectue une remontée depuis la case $m[n1, n2]$. À partir de la case $m[i, j]$, on ajoute $s1[i-1]$ à `lst` lorsque

$$m[i, j] - 1 = m[i-1, j] = m[i, j-j] = m[i-1, j-1]$$

Exemple A partir de la table ci-dessus, l'évolution de `lst` peut être :

```
i= 10, j = 12 ,lst = []
i= 10, j = 11 ,lst = []
i= 10, j = 10 ,lst = []
i= 9, j = 9 ,lst = ['j']
i= 8, j = 9 ,lst = ['j']
i= 8, j = 8 ,lst = ['j']
i= 8, j = 7 ,lst = ['j']
i= 7, j = 6 ,lst = ['j', 'h']
i= 7, j = 5 ,lst = ['j', 'h']
i= 7, j = 4 ,lst = ['j', 'h']
i= 6, j = 3 ,lst = ['j', 'h', 'g']
i= 5, j = 3 ,lst = ['j', 'h', 'g']
i= 4, j = 2 ,lst = ['j', 'h', 'g', 'e']
i= 3, j = 2 ,lst = ['j', 'h', 'g', 'e']
i= 2, j = 1 ,lst = ['j', 'h', 'g', 'e', 'c']
i= 1, j = 1 ,lst = ['j', 'h', 'g', 'e', 'c']
i= 0, j = 1 ,lst =['j', 'h', 'g', 'e', 'c']
'ceghj
```

Question 7 : Écrire une fonction `afficher_seq_commune` prenant en entrée deux chaînes $s1$ et $s2$ et retournant la plus longue sous-chaîne commune de $s1$ et $s2$.