Devoir surveillé n°2 - 2h - Jeudi 16 octobre 2025

Rédiger sur une copie double. La calculatrice n'est pas autorisée.

Consignes de présentation :

- *Nom Prénom lisibles en haut à gauche, classe en haut à droite, DS n°1 au milieu et encadré à la règle.*
- *Vous laissez ensuite 5 ou 6 centimètres pour mes éventuels commentaires, délimités par deux traits tracés à la règle.*
- *S'il n'y a pas de marge déjà tracée sur votre feuille, vous en tracez une à 4 cm du bord de la feuille.*
- ***Les codes Python demandés doivent être écrit très lisiblement, avec une indentation très claire.***

Exercice 1 : Voici le code d'une fonction, qui prend en argument une liste L et un entier naturel p inférieur strictement à la taille de la liste :

```
1 def mystere(L,p):
2     assert p>=0 and p<=len(L)-1
3     if p==len(L)-1:
4         return True
5     if L[p]>L[p+1]:
6         return False
7     else:
8         return mystere(L,p+1)
```

1. Soit `L=[6,9,4,8,12]`. Que retournent `mystere(L,2)` et `mystere(L,0)` ? Donner la liste des appels récursifs dans chacun des cas.
2. Que fait cette fonction dans le cas général ? *Vous donnerez votre réponse avec une phrase claire.*
3. Écrire une version non récursive de cet algorithme.

Exercice 2 : .

1. (a) Écrire une fonction `occurrences(liste)` qui prend en argument une liste de nombres et renvoie un dictionnaire, dont les clés sont les différents nombres de la liste avec pour valeur le nombre d'occurrences de chaque nombre.
   Par exemple `occurrences([50, 10, 30, 40, 10])` renvoie le dictionnaire {`10:2,50:1,30:1,40:1`}.
   (b) Déterminer en fonction de la taille n de la liste, la complexité de cette fonction `occurrences`.
2. On dispose de deux listes de nombres qui sont tous de même type (soit du type int, soit du type float).
   On suppose de plus que ces deux listes sont de même taille. Le but de cette question est de tester si les deux listes contiennent les mêmes nombres (pas nécessairement dans le même ordre).
   (a) Écrire une fonction `compare(liste1,liste2)` qui prend en paramètres deux listes de nombres de même longueur et renvoie `True` si les deux listes contiennent les mêmes nombres, pas nécessairement dans le même ordre, et `False` sinon. Cette fonction devra impérativement utiliser la fonction `occurrences`.
   Exemple : Soit les listes suivantes :

```
1 liste1 = [10, 20, 30, 40, 50]
2 liste2 = [50, 75, 30, 20, 40]
3 liste3 = [50, 20, 30, 40, 10]
```

   L'appel à `compare(liste1,liste2)` devra renvoyer `False`, et l'appel à `compare(liste1,liste3)` devra renvoyer `True`.

(b) Si on note n la taille commune des liste `L1` et L2, déterminer, en fonction de n, la complexité de la fonction `compare` (vous veillerez à justifier l'obtention de cette complexité).

Exercice 3 : On considère une matrice carrée M de taille $n \times n$ contenant des entiers positifs ou négatifs :

$$M = \begin{pmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \dots & m_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1,0} & m_{n-1,1} & \dots & m_{n-1,n-1} \end{pmatrix}$$

On souhaite se déplacer du coin supérieur gauche $(0,0)$ jusqu'au coin inférieur droit $(n-1, n-1)$.
À chaque étape, on peut **uniquement aller vers la droite ou aller vers le bas**. (pas de déplacement "en diagonale")
Chaque case traversée ajoute son poids (valeur du coefficient) à la somme totale du chemin.

Le but est de déterminer le **poids maximal possible** obtenu en suivant un chemin valide de $(0,0)$ à $(n-1, n-1)$.

1. **Exemple :** On considère la matrice suivante $M = \begin{pmatrix} 5 & 3 & 2 \\ 1 & 9 & 1 \\ 0 & 2 & 8 \end{pmatrix}$

   Un exemple de chemin valide est $5 \to 1 \to 9 \to 1 \to 8$, son poids total est $5+1+9+1+8 = 24$.

   Répertorier tous les chemins valides possibles de la case $(0,0)$ à la case $(2,2)$, et calculer leur poids total. Quel est le poids maximal obtenu ?

2. **Recherche exhaustive***(un peu de maths)*
   (a) Montrer que le nombre de chemins distincts possibles pour aller de $(0,0)$ à $(n-1, n-1)$ est le coefficient binomial $\binom{2(n-1)}{n-1}$.
   (b) En utilisant la formule de Stirling $n! \sim \sqrt{2\pi n} \times \dfrac{n^n}{e^n}$, déterminer un équivalent de $\binom{2(n-1)}{n-1}$, et en déduire la classe de complexité d'un algorithme qui consisterait à calculer les poids totaux de tous les chemins possibles pour ensuite prendre le maximum de ces poids.

3. **Une méthode efficace**
   (a) *Préliminaire :* Écrire une fonction `max2`, d'argument deux nombres entiers ou flottants, qui retourne le maximum de ces deux nombres.

   On va coder de façon récursive (avec mémoïsation) une fonction `poids_max(M,i,j)` qui, pour un indice de ligne i et un indice de colonne j, va renvoyer le poids maximum des chemins possibles de la case (i,j) à la case $(n-1, n-1)$. (n étant la taille de la matrice carrée M, que l'on suppose codée comme une liste de liste)

   (b) Si $(i,j) = (n-1, n-1)$, que doit renvoyer `poids_max(M,i,j)` ?
   (c) Si $i = n-1$ et $j \neq n-1$, expliquer pourquoi `poids_max(M,i,j)=M[i][j]+poids_max(M,i,j+1)`.
   (d) Si $i \neq n-1$ et $j = n-1$, que doit renvoyer `poids_max(M,i,j)` ?
   (e) Si $i \neq n-1$ et $j \neq n-1$, donner la valeur de `poids_max(M,i,j)` en fonction de `M[i][j]`, de `poids_max(M,i,j+1)` et de `poids_max(M,i+1,j)`. *(on pourra utiliser la fonction `max2` créée à la question (a))*
   (f) Ecrire le script permettant de coder la fonction `poids_max(M,i,j)` : cette fonction devra être **récursive**, et **utiliser un dictionnaire** pour memoïser les résultats obtenus.
   (g) Préciser avec quels arguments on doit appeler la fonction `poids_max` pour obtenir le poids maximal cherché sur une matrice M (c'est-à-dire en allant de $(0,0)$ à $(n-1, n-1)$).