Mercredi 10 décembre 2025 – *Durée : 2 heures*

Devoir Surveillé n°1

L'usage de la calculatrice est INTERDIT.

L'épreuve est à traiter en langage Python, sauf les questions sur les bases de données qui seront traitées en langage SQL.

Les différents algorithmes doivent être rendus dans leur forme définitive sur la copie en respectant les éléments de syntaxe du langage (les brouillons ne sont pas acceptés).

Il est demandé au candidat de bien vouloir rédiger ses réponses en précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions.

La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés.

Exercice n°1 Mise en jambe pythonesque *(Durée ∼ 20 min)*

L'utilisation de `max`, `min`, `L.index()`, `L.sorted`, `d.keys`, `d.items`, `d.value`, et des fonctions de la bibliothèque `numpy` est interdite dans cet exercice.

Q1. Écrire la fonction `maximum(u:list)->(float,int)` prenant en entrée une liste `u` d'entiers (ou flottants), et renvoyant sa valeur maximale, et le premier indice de ce maximum.

Q2. Écrire une fonction `occurrences(texte:str)->dict` qui prend en argument une chaîne de caractère `texte` et renvoie un dictionnaire dont les clés sont les lettres qui apparaissent dans le texte et les valeurs le nombre d'occurrences de ces lettres.

Par exemple : `occurrences('ACCTAGCCCTA')` renverra `{'A':3,'C':5,'T':2,'G':1}`.

On souhaite écrire un algorithme permettant de calculer $\binom{n}{k}$. Pour cela, on utilise la formule de récurrence suivante :

$$\binom{n}{k} = \begin{cases} 1 & \text{si } k = n \text{ ou } k = 0 \\ n & \text{si } k = 1 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{sinon} \end{cases}$$

On souhaite procéder de bas en haut en **stockant** les valeurs successives **dans un tableau à deux dimensions** en le complétant de bas à haut, c'est-à-dire des petites valeurs aux plus grandes valeurs.

L'élément colonne $i \in [\![0, k]\!]$ ligne $j \in [\![0, n]\!]$ contient le coefficient $\binom{j}{i}$.

Q3. Compléter à la main le tableau **sur le document réponse** pour calculer $\binom{5}{3}$.

Q4. Dans quelle case du tableau se trouve le résultat de $\binom{5}{3}$? Généraliser pour $\binom{n}{k}$.

Q5. Écrire une fonction `tableau_nul(p:int,m:int)->list[list]` qui renvoie un tableau de `p` lignes et `m` colonnes zéros.

Q6. Compléter la fonction `Pascal(k:int,n:int)->int` **sur le document réponse** qui complète le tableau permettant de calculer le coefficient $\binom{n}{k}$

Exercice n°2 Festivals de musique *(Durée ∼ 1h40)*

L'objectif de ce sujet est d'étudier les festivals de musique :

— Partie I : étudie la préparation de son programme de festival en utilisant l'algorithme glouton ;

— Partie II : étudie la préparation de son programme du festival en utilisant la programmation dynamique ;

— Partie III : traite d'une base de données ;

Vous pouvez utiliser ultérieurement une fonction que vous l'ayez ou non écrite.

Partie I Réalisation de son programme du festival

Vous vous rendez dans un festival de musique.

Le concert d'indice i est défini par l'instant de début (d_i), l'instant de fin (f_i) et son intérêt (à qui vous lui avez attribué une valeur v_i) : $[d_i, f_i, v_i]$.

Ainsi la liste des n concerts est notée $C = [[d_0, f_0, v_0], ..., [d_{n-1}, f_{n-1}, v_{n-1}]]$.

Vous souhaitez assister à un maximum de concerts tout en optimisant les intérêts que vous y portez. On cherche à déterminer le sous-ensemble S de C tel que $\sum_{i \in S} v_i$ est maximal.

Vous ne pouvez bien évidemment assister qu'à un concert à la fois, et vous n'assistez qu'à des concerts entiers. L'intersection entre les intervalles de deux concerts auxquels vous assistez est l'ensemble vide ou limité à un nombre $(d_i = f_j)$.

A - Questions préliminaires

Q1. Écrire une fonction `valeur(C:list)->int` qui prend en argument une liste `C` du format précédent (représentant un ensemble de concerts) et qui renvoie la valeur totale de l'ensemble des concerts.

On souhaite trier les concerts par heure de fin croissante en adaptant l'**algorithme du tri rapide**.

Principe : On choisit un élément de la liste, le premier élément de la liste, que l'on notera `p`, appelé pivot, on partitionne la liste en deux sous listes : une liste contenant les concerts d'heure de fin strictement inférieure à celle de `p`, et une liste contenant les concerts d'heure de fin strictement supérieure à celle de `p`. On trie récursivement chacune des deux listes et on rassemble le tout.

Q2. **Compléter sur le document réponse** la fonction `tri_rapide(C:list[list])->list[list]` qui prend en argument une liste `C` du type de la liste `concerts` que l'on souhaite trier par ordre d'instants de fin croissant.

Pour toute la suite, on suppose que la liste `C` est triée par ordre d'instant de fin croissant.

Q3. Écrire une fonction `compatible(C:list[list],k:int)->int` qui détermine l'indice du concert (parmi la liste `C`) compatible avec le concert d'indice `k` donné et qui se termine au plus près de ce concert `k`.

La fonction renvoie -1 si aucun concert n'est compatible avec le concert k.

B - Algorithme glouton

On souhaite envisager une solution exploitant un algorithme glouton.

On suppose qu'il n'y a pas deux concerts qui finissent à la même heure.

Pour la suite, on suppose que la liste `C` est triée par ordre d'instant de fin croissant.

On donne l'exemple de la liste des concerts suivante :

`concerts=[[9,10,3],[9,13,2],[11,14,1],[11,15,3],[17,21,3],[19,22,2]]`.

Le principe de l'algorithme glouton mis en œuvre ici est le suivant : on choisit le concert se terminant le plus tard, puis le concert se terminant au plus tard parmi ceux qui sont compatibles avec le premier, et ainsi de suite.

Q4. Mettre en œuvre l'algorithme glouton à la main pour déterminer le programme des concerts optimal. Quelle est la valeur totale de ce programme ? Est-ce que la solution trouvée avec l'algorithme glouton est optimale ? Peut-on trouver une meilleure solution ?

Q5. Compléter **sur le document réponse** la fonction `glouton(C:list[list])->list` qui prend en argument une liste `C` des concerts sous le format défini précédemment, et renvoie la liste `Cavoir` des concerts à aller voir en utilisant l'algorithme glouton décrit précédemment.

On pourra avantageusement utiliser la fonction `compatible` écrite précédemment.

Pour la liste donnée en exemple, `glouton(concerts)` renvoie `[[19,22,2],[11,15,3],[9,10,3]]`

Partie II Programmation dynamique

Vous avez des préférences en terme de concerts, et vous avez donc attribué une valeur (niveau de satisfaction) à chaque concert. Vous souhaitez maximiser la valeur totale, c'est-à-dire déterminer le sous-ensemble D de C tel que $\sum_{i \in D} v_i$ est maximal.

On suppose qu'on a une liste des concerts du type `concerts=[[d0,f0,v0],...]` triés par heure de fin croissante.

On note $S(C,k)$ la solution au problème, c'est-à-dire la somme maximale des valeurs des concerts qu'on peut aller voir, quand on considère les concerts $C_0 ... C_k$ jusqu'au rang k inclus.

Q6. Pour quelle valeur de k obtient-on la solution à notre problème ?

On donne la relation de récurrence : $S(C,k) = \begin{cases} 0 & \text{si } k = -1 \\ \max\Big(v_k + S\big(C, \text{compatible}(C,k)\big), S(C,k-1)\Big) & \text{sinon} \end{cases}$

Q7. Expliquer la valeur de $S(C,-1)$.

Q8. Expliquer précisément la relation de récurrence pour $k \neq -1$.

Q9. Proposer une fonction récursive « naïve » `sol_rec(C:list[list],k:int)->int` à partir de la relation de récurrence, qui renvoie la valeur totale maximale que l'on peut obtenir.

Q10. Quel est le problème de la fonction précédente ? Pourquoi est-il intéressant d'utiliser la programmation dynamique ?

Q11. On souhaite améliorer la fonction récursive précédente en utilisant la mémoïsation.

Compléter sur le document réponse la fonction récursive `sol_mem(C:list[list])->int` .

Partie III Base de données des festivals

On considère une base de données qui regroupe les informations concernant les festivals qui ont lieu tout au long de l'année, les groupes et musiciens y participant.

On considère la base de données suivante :

`Festivals` (`id`, `nom`, `ville`, `tarif`, `debut`, `fin`)

`Concerts` (`id`, `idgroupe`, `idfest`, `date`)

`Groupes` (`id`, `nom`)

`Musiciens` (`id`, `nom`, `prénom`, `nationalité`, `idgroupe`)

Les attributs `id` sont des entiers. L'attribut `tarif` est un flottant. Les attributs `date`, `debut` (date de début), `fin` (date de fin) sont des entiers du format `aaaammjj` (par exemple pour aujourd'hui : 20251210). Les autres attributs sont des chaînes de caractères.

`idgroupe` est une clé étrangère des tables `Musiciens` et `Concerts` en relation avec la clé primaire `id` de la table `Groupes`. `idfest` de la table `Concerts` est une clé étrangère en relation avec la clé primaire `id` de la table `Festivals`.

Q12. Écrire la requête SQL qui renvoie la liste des noms des festivals ayant lieux entièrement entre le 20 juillet 2026 (fin des concours) et le 31 août 2026 (entrée en école) classés par tarif décroissant.

Q13. Écrire la requête SQL qui renvoie le nombre de musiciens pour chaque nationalité.

Q14. Écrire la requête SQL qui renvoie le nom du troisième et quatrième festival les moins chers.

Q15. Écrire la requête SQL qui renvoie la liste des musiciens (nom, prénom) du groupe « Quatuor Debussy ».

Q16. Écrire la requête SQL qui renvoie la liste des noms des groupes participants au festival « Les vieilles charrues ».

Q17. Écrire la requête SQL qui renvoie les festivals proposant au moins 15 concerts.

Q18. Écrire la requête SQL qui renvoie le pourcentage de festivals de prix supérieur au prix moyen de l'ensemble des festivals.

DOCUMENT RÉPONSE À RENDRE

NOM : _____ Prénom : _____

Q3

j \ i	0	1	2	3
0				
1				
2				
3				
4				
5				

Q6

```
def Pascal(k,n):
    tab = ..........................................  # initialisation du tableau
    for j in range( ..... , ..... ) : # remplissage de la première colonne
        tab[.....][.....] = ..................
    for j in range( ..... , ..... ) : # remplissage de la diagonale
        tab[.....][.....] = ..................
    for j in range( ..... , ..... ) : # remplissage des autres lignes
        for i in range( ..... , ..... ) : # remplissage des autres colonnes
            tab[.....][.....] =  ..................
    return ..................
```

Q2

```
def tri_rapide(C):
    if len(C)<=1 : # cas de base
        return ..........
    p=C[0] # choix du pivot, le premier élément de C
    Cinf,Csup=[],[] # Cinf (Csup) liste des objets de C de valeurs inférieures (supérieures) à la valeur du pivot
    for  ............................ : # parcours des éléments de C
        if ............................ :
            ..............................
        else :
            ..............................
    return ..................... + [p] + ..................... # appels récursifs et concaténation
```

Q5

```
def glouton(C):
    Cavoir = [] # liste des concerts à voir
    i = ..... # rang du concert terminant le plus tard
    .....   # ajout à Cavoir du concert terminant le plus tard
    i = ..... # concert précédent compatible
    while ..... : # tant qu'il y a un concert compatible
        ..... # ajout du concert i
        ..... # recherche d'un concert précédent compatible
    return .....
```

Q11

```
def sol_mem(C):
    """ renvoie la valeur totale maximales concerts de C pouvant être vus"""
    dico = {-1:0} # variable locale pour sol_mem, globale pour mem
    def mem(i):
        """ Fonction récursive qui calcule S(C,i) """
        if ..... : # la solution a déjà été calculée
            return .....
        else:
            vi = ..... # valeur du concert  i
            a = .....  #  si le concert i ne fait pas partie de la solution optimale
            b = .....  #  si le concert i fait partie de la solution optimale
            s = ..... # S(C,i) est la solution optimale parmi les deux précédentes
            dico[i]= ..... # on mémoïse
        return ..... # renvoie S(C,i)
    return ..... # appel de mem au rang ... pour avoir la solution du problème
```