NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : 2 heures

L'usage de la calculatrice n'est pas autorisé

Le candidat traite au choix 3 exercices parmi les 4 exercices proposés

Chaque exercice devra être traité sur une copie séparée

Exercice 1

Bases de données

Dans cet exercice, on pourra utiliser les mots clés suivants du langage SQL :

```
SELECT, FROM, WHERE, JOIN, ON, INSERT INTO, VALUES
MIN, MAX, OR, AND
```

Les fonctions d'agrégation `MIN(propriete)` et `MAX(propriete)` renvoient respectivement la plus petite et la plus grande valeur de l'attribut `propriete` pour les enregistrements sélectionnés.

Des acteurs ayant joué dans différentes pièces de théâtre sont recensés dans une base de données Theatre dont le schéma relationnel est donné ci-dessous :

- **Piece**(idPiece, titre, langue)
- **Acteur**(idActeur, nom, prenom, anneeNaiss)
- **Role**(idPiece, idActeur, nomRole)

Dans ce schéma, les clés primaires sont soulignées en trait plein et les clés étrangères en pointillés.
L'attribut `idPiece` de la relation `Role` est une clé étrangère faisant référence à l'attribut `idPiece` de la relation `Piece`.
L'attribut `idActeur` de la relation `Role` est une clé étrangère faisant référence à l'attribut `idActeur` de la relation `Acteur`.
Tous les attributs dont le nom est préfixé par `id` sont des nombres entiers ainsi que l'attribut `anneeNaiss`. Les autres attributs sont des chaînes de caractères.

1. Expliquer pourquoi il n'est pas possible d'insérer une entrée dans la relation `Role` si les relations `Piece` et `Acteur` sont vides.

2. Dans la pièce « Le Tartuffe », l'acteur Micha Lescot a joué le rôle de Tartuffe.
   L'identifiant de Micha Lescot est 389761 et celui de cette pièce est 46721.
   Écrire une requête SQL permettant d'ajouter ce rôle dans la table (ou relation) `Role`.

3. Expliquer ce que fait la requête SQL suivante.

Code SQL

```sql
UPDATE Piece
SET langue = "Anglais"
WHERE langue = "Américain" OR langue = "Britannique";
```

4. Pour chacun des quatre items suivants, écrire une requête SQL permettant d'extraire les informations demandées.

   a. Le nom et prénom des artistes nés après 1990.

   b. L'année de naissance du plus jeune artiste.

   c. Le nom des rôles joués par l'acteur Vincent Macaigne.

   d. Le titre des pièces écrites en Russe dans lesquelles l'actrice Jeanne Balibar a joué.

Exercice 2

Piles et POO

On crée une classe `Pile` qui modélise la structure d'une pile d'entiers.
Le constructeur de la classe initialise une pile vide.
La définition de cette classe sans l'implémentation de ses méthodes est donnée ci-dessous.

Code Python

```
class Pile:
    def __init__(self):
        """Initialise la pile comme une pile vide."""

    def est_vide(self):
        """Renvoie True si la liste est vide, False sinon."""

    def empiler(self, e):
        """Ajoute l'élément e sur le sommet de la pile,
        ne renvoie rien."""

    def depiler(self):
        """Retire l'élément au sommet de la pile et le renvoie."""

    def nb_elements(self):
        """Renvoie le nombre d'éléments de la pile. """

    def afficher(self):
        """Affiche de gauche à droite les éléments de la pile, du
            fond
        de la pile vers son sommet. Le sommet est alors l'élément
       affiché le plus à droite. Les éléments sont séparés par une
        virgule. Si la pile est vide la méthode affiche « pile
        vide »."""
```

Seules les méthodes de la classe ci-dessus doivent être utilisées pour manipuler les objets de la classe `Pile`.

1. a. Écrire une suite d'instructions permettant de créer une instance de la classe `Pile` affectée à une variable `pile1` contenant les éléments 7, 5 et 2 insérés dans cet ordre. Ainsi, à l'issue de ces instructions, l'instruction `pile1.afficher()` produit l'affichage : 7, 5, 2.

   b. Donner l'affichage produit après l'exécution des instructions suivantes.

```
element1 = pile1.depiler()
pile1.empiler(5)
pile1.empiler(element1)
pile1.afficher()
```

2. On donne la fonction `mystere` suivante :

Code Python

```
def mystere(pile, element):
    pile2 = Pile()
    nb_elements = pile.nb_elements()
    for i in range(nb_elements):
        elem = pile.depiler()
        pile2.empiler(elem)
        if elem == element:
            return pile2
    return pile2
```

   a. Dans chacun des quatre cas suivants, quel est l'affichage obtenu dans la console ?

   - cas n°1

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 2).afficher()
```

   - cas n°2

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 9).afficher()
```

- cas n°3

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 3).afficher()
```

- cas n°4

```
>>>pile.est_vide()
True
>>>mystere(pile, 3).afficher()
```

b. Expliquer ce que permet d'obtenir la fonction `mystere`.

3. Écrire une fonction `etendre(pile1, pile2)` qui prend en arguments deux objets `Pile` appelés `pile1` et `pile2` et qui modifie `pile1` en lui ajoutant les éléments de `pile2` rangés dans l'ordre inverse.
Cette fonction ne renvoie rien.
On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>>pile1.afficher()
7, 5, 2, 3
>>>pile2.afficher()
1, 3, 4
>>>etendre(pile1, pile2)
>>>pile1.afficher()
7, 5, 2, 3, 4, 3, 1
>>>pile2.est_vide()
True
```

4. Écrire une fonction `supprime_toutes_occurences(pile, element)` qui prend en arguments un objet `Pile` appelé `pile` et un élément `element` et supprime tous les éléments `element` de `pile`.
On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>>pile.afficher()
7, 5, 2, 3, 5
>>>supprime_toutes_occurences (pile, 5)
>>>pile.afficher()
7, 2, 3
```

Exercice 3

Diviser pour régner

Dans un tableau PYTHON d'entiers `tab`, on dit que le couple d'indices `(i, j)` forme une inversion lorsque `i < j` et `tab[i] > tab[j]`. On donne ci-dessous quelques exemples.

- Dans le tableau `[1, 5, 3, 7]`, le couple d'indices `(1, 2)` forme une inversion car `5 > 3`.
  Par contre, le couple `(1, 3)` ne forme pas d'inversion car `5 < 7`.
  Il n'y a qu'une inversion dans ce tableau.
- Il y a trois inversions dans le tableau `[1, 6, 2, 7, 3]`, à savoir les couples d'indices `(1, 2)`, `(1, 4)` et `(3, 4)`.
- On peut compter six inversions dans le tableau `[7, 6, 5, 3]` : les couples d'indices `(0, 1)`, `(0, 2)`, `(0, 3)`, `(1, 2)`, `(1, 3)` et `(2, 3)`.

On se propose dans cet exercice de déterminer le nombre d'inversions dans un tableau quelconque.

Questions préliminaires

1. Expliquer pourquoi le couple `(1, 3)` est une inversion dans le tableau `[4, 8, 3, 7]`.
2. Justifier que le couple `(2, 3)` n'en est pas une.

Partie A : Méthode itérative

Le but de cette partie est d'écrire une fonction itérative `nombre_inversions` qui renvoie le nombre d'inversions dans un tableau.
Pour cela, on commence par écrire une fonction `fonction1` qui sera ensuite utilisée pour écrire la fonction `nombre_inversions`.

1. On donne la fonction suivante.

Code Python

```
def fonction1(tab, i):
    nb_elem = len(tab)
    cpt = 0
    for j in range(i+1, nb_elem):
        if tab[j] < tab[i]:
        cpt += 1
    return cpt
```

a. Indiquer ce que renvoie `fonction1(tab, i)` dans les cas suivants.

- cas n°1 : `tab = [1, 5, 3, 7]` et `i = 0`.
- cas n°2 : `tab = [1, 5, 3, 7]` et `i = 1`.
- cas n°3 : `tab = [1, 5, 2, 6, 4]` et `i = 1`.

b. Expliquer ce que permet de déterminer cette fonction.

2. En utilisant la fonction précédente, écrire une fonction `nombre_inversions(tab)` qui prend en argument un tableau et renvoie le nombre d'inversions dans ce tableau.
On donne ci-dessous les résultats attendus pour certains appels.

```
>>> nombre_inversions([1, 5, 7])
0
>>> nombre_inversions([1, 6, 2, 7, 3])
3
>>> nombre_inversions([7, 6, 5, 3])
6
```

3. Quel est l'ordre de grandeur de la complexité en temps de l'algorithme obtenu ?
Aucune justification n'est attendue.

Partie B : Méthode récursive

Le but de cette partie est de concevoir une version récursive de la fonction de la partie précédente. On définit pour cela des fonctions auxiliaires.

1. Donner le nom d'un algorithme de tri ayant une complexité meilleure que quadratique.

Dans la suite de cet exercice, on suppose qu'on dispose d'une fonction `tri(tab)` qui prend en argument un tableau et renvoie un tableau contenant les mêmes éléments rangés dans l'ordre croissant.

2. Écrire une fonction `moitie_gauche(tab)` qui prend en argument un tableau `tab` et renvoie un nouveau tableau contenant la moitié gauche de `tab`. Si le nombre d'éléments de `tab` est impair, l'élément du centre se trouve dans cette partie gauche. On donne ci-dessous les résultats attendus pour certains appels.

```
>>> moitie_gauche([])
[]
>>> moitie_gauche([4, 8, 3])
[4, 8]
>>> moitie_gauche ([4, 8, 3, 7])
[4, 8]
```

Dans la suite, on suppose qu'on dispose de la fonction `moitie_droite(tab)` qui renvoie la moitié droite sans l'élément du milieu

3. On suppose qu'une fonction `nb_inv_tab(tab1, tab2)` a été écrite.
Cette fonction renvoie le nombre d'inversions du tableau obtenu en mettant bout à bout les tableaux `tab1` et `tab2`, à condition que `tab1` et `tab2` soient triés dans l'ordre croissant.
On donne ci-dessous deux exemples d'appel de cette fonction :

```
>>> nb_inv_tab([3, 7, 9], [2, 10])
3
>>> nb_inv_tab([7, 9, 13], [7, 10, 14])
3
```

En utilisant la fonction `nb_inv_tab` et les questions précédentes, écrire une fonction récursive `nb_inversions_rec(tab)` qui permet de calculer le nombre d'inversions dans un tableau.
Cette fonction renverra le même nombre que `nombre_inversions(tab)` de la partie A.
On procédera de la façon suivante :

- Séparer le tableau en deux tableaux de tailles égales (à une unité près).
- Appeler récursivement la fonction `nb_inversions_rec` pour compter le nombre d'inversions dans chacun des deux tableaux.
- Trier les deux tableaux (on rappelle qu'une fonction de tri est déjà définie).
- Ajouter au nombre d'inversions précédemment comptées le nombre renvoyé par la fonction `nb_inv_tab` avec pour arguments les deux tableaux triés.

Exercice 4

Gestion des processus

1. Les états possibles d'un processus sont : *prêt*, *élu*, *terminé* et *bloqué*.

   a. Expliquer à quoi correspond l'état élu.

   b. Proposer un schéma illustrant les passages entre les différents états.

2. On suppose que quatre processus C_1, C_2, C_3 et C_4 sont créés sur un ordinateur, et qu'aucun autre processus n'est lancé sur celui-ci, ni préalablement ni pendant l'exécution des quatre processus.

   L'ordonnanceur, pour exécuter les différents processus prêts, les place dans une structure de données de type file. Un processus prêt est enfilé et un processus élu est défilé.

a. Parmi les propositions suivantes, recopier celle qui décrit le fonctionnement des entrées/sorties dans une file :

   - Premier entré, dernier sorti
   - Premier entré, premier sorti
   - Dernier entré, premier sorti

b. On suppose que les quatre processus arrivent dans la file et y sont placés dans l'ordre C_1, C_2, C_3 et C_4.

   - Les temps d'exécution totaux de C_1, C_2, C_3 et C_4 sont respectivement 100 ms, 150 ms, 80 ms et 60 ms.
   - Après 40 ms d'exécution, le processus C_1 demande une opération d'écriture disque, opération qui dure 200 ms. Pendant cette opération d'écriture, le processus C_1 passe à l'état bloqué.
   - Après 20 ms d'exécution, le processus C_3 demande une opération d'écriture disque, opération qui dure 10 ms. Pendant cette opération d'écriture, le processus C_3 passe à l'état bloqué.

   Sur la frise chronologique donnée en annexe (à rendre avec la copie), les états du processus C_2 sont donnés. Compléter la frise avec les états des processus C1, C3 et C4.

3. On trouvera ci- dessous deux programmes rédigés en pseudo-code.

Verrouiller un fichier signifie que le programme demande un accès exclusif au fichier et l'obtient si le fichier est disponible.

Programme 1	Programme 2
Verrouiller fichier1	Verrouiller fichier2
Calculs sur fichier1	Verrouiller fichier1
Verrouiller fichier2	Calculs sur fichier1
Calculs sur fichier1	Calculs sur fichier2
Calculs sur fichier2	Déverrouiller fichier1
Calculs sur fichier1	Déverrouiller fichier2
Déverrouiller fichier2	
Déverrouiller fichier1	

a. En supposant que les processus correspondant à ces programmes s'exécutent simultanément (exécution concurrente), expliquer le problème qui peut être rencontré.

b. Proposer une modification du programme 2 permettant d'éviter ce problème.