

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2022

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 1

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 16 pages numérotées de 1/16 à 16/16.

Le candidat traite au choix 3 exercices parmi les 5 exercices proposés

Chaque exercice est noté sur 4 points.

EXERCICE 1 (4 points)

Cet exercice porte sur les structures de données (listes, piles et files).

On cherche ici à mettre en place des algorithmes qui permettent de modifier l'ordre des informations contenues dans une file. On considère pour cela les structures de données abstraites de Pile et File définies par leurs fonctions primitives suivantes :

Pile :

- `creer_pile_vide()` renvoie une pile vide ;
- `est_pile_vide(p)` renvoie `True` si la pile `p` est vide, `False` sinon ;
- `empiler(p, element)` ajoute `element` au sommet de la pile `p` ;
- `depiler(p)` renvoie l'élément se situant au sommet de la pile `p` en le retirant de la pile `p` ;
- `sommet(p)` renvoie l'élément se situant au sommet de la pile `p` sans le retirer de la pile `p`.

File :

- `creer_file_vide()` renvoie une file vide ;
- `est_file_vide(f)` renvoie `True` si la file `f` est vide, `False` sinon ;
- `enfiler(f, element)` ajoute `element` dans la file `f` ;
- `defiler(f)` renvoie l'élément à la tête de la file `f` en le retirant de la file `f`.

On considère de plus que l'on dispose d'une fonction permettant de connaître le nombre d'éléments d'une file :

- `taille_file(f)` renvoie le nombre d'éléments de la file `f`.

On représentera les files par des éléments en ligne, l'élément de droite étant la tête de la file et l'élément de gauche étant la queue de la file. On représentera les piles en colonnes, le sommet de la pile étant le haut de la colonne.

La file suivante est appelée `f` :

4	3	8	2	1
---	---	---	---	---

La pile suivante est appelée `p` :

5
8
6
2

1. Les quatre questions suivantes sont indépendantes. Pour chaque question, on repartira de la pile `p` et de la file `f` initiales (présentées ci-dessus)

a. Représenter la file `f` après l'exécution du code suivant.

```
enfiler(f, defiler(f))
```

b. Représenter la pile `p` après l'exécution du code suivant.

```
empiler(p, depiler(p))
```

c. Représenter la pile `p` et la file `f` après l'exécution du code suivant.

```
for i in range(2):  
    enfiler(f, depiler(p))
```

d. Représenter la pile p et la file f après l'exécution du code suivant.

```
for i in range(2):
    empiler(p, defiler(f))
```

2. On donne ici une fonction `mystere` qui prend une file en argument, qui modifie cette file, mais qui ne renvoie rien.

```
def mystere(f):
    p = creer_pile_vide()
    while not est_file_vide(f):
        empiler(p, defiler(f))
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
    return p
```

Préciser l'état de la variable f après chaque boucle de la fonction

`mystere` appliquée à la file

1	2	3	4
---	---	---	---

 Indiquer le contenu de la pile renvoyée par la fonction.

3. On considère la fonction `knuth(f)` suivante dont le paramètre est une file :

```
def knuth(f):
    p=creer_pile_vide()
    N=taille_file(f)
    for i in range(N):
        if est_pile_vide(p):
            empiler(p, defiler(f))
        else:
            e = defiler(f)
            if e >= sommet(p):
                empiler(p, e)
            else:
                while not est_pile_vide(p) and e < sommet(p):
                    enfiler(f, depiler(p))
                empiler(p, e)
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
```

a. Recopier et compléter le tableau ci-dessous qui détaille le fonctionnement de cet algorithme étape par étape pour la file 2, 1, 3. Une étape correspond à une modification de la pile ou de la file. Le nombre de colonnes peut bien sûr être modifié.

f	2,1,3	2,1										
p	<input type="text"/>	<input type="text" value="3"/>	<input type="text"/>									

b. Que fait cet algorithme ?

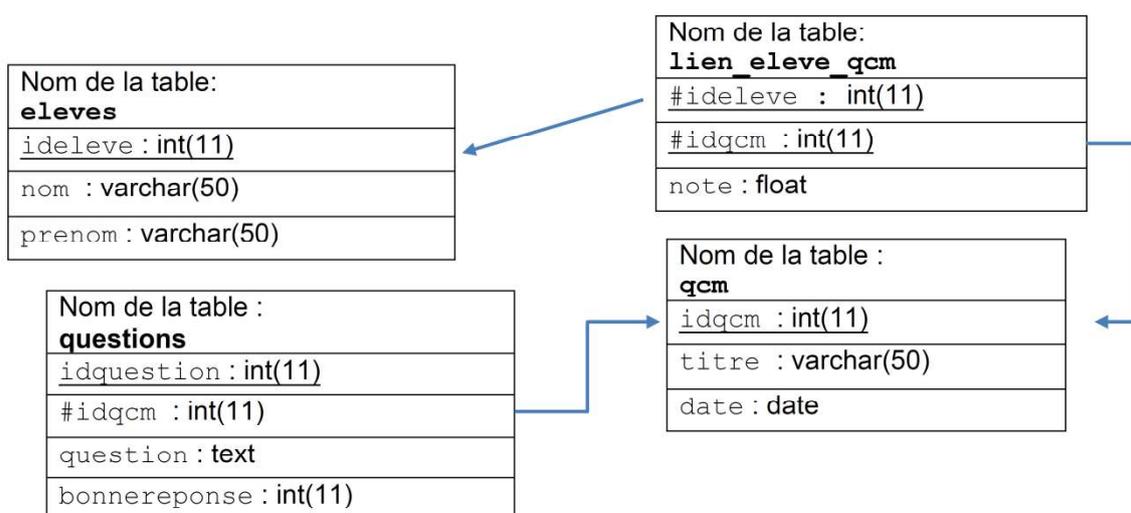
EXERCICE 3 (4 points)

Cet exercice porte sur les bases de données (bases de données relationnelles, langage SQL).

Un rappel sur la syntaxe de quelques fonctions SQL est donné en annexe 1 en fin de sujet.

Un enseignant a mis en place un site web qui permet à ses élèves de faire des QCM (questionnaire à choix multiples) de NSI en ligne.

L'enseignant a créé une base de données nommée QCM_NSI pour gérer ses QCM, contenant les quatre relations (appelé aussi communément "table") du schéma relationnel ci-dessous :



Dans le schéma relationnel précédent, un attribut souligné indique qu'il s'agit d'une clé primaire. Un attribut précédé du symbole # indique qu'il s'agit d'une clé étrangère et la flèche associée indique l'attribut référencé. Ainsi, par exemple, l'attribut ideleve de la relation `lien_eleve_qcm` est une clé étrangère qui fait référence à l'attribut ideleve de la relation `eleves`.

Dans le cas de la relation `lien_eleve_qcm` la clé primaire est composée de l'association des deux attributs ideleve et idqcm, eux-mêmes étant des clés étrangères.

On donne ci-dessous le contenu exhaustif des relations :

Table eleves

ideleve	nom	prenom
2	Dubois	Thomas
3	Dupont	Cassandra
4	Marty	Mael
5	Bikila	Abebe

Table lien_eleve_qcm

ideleve	idqcm	note
2	1	12
2	3	18
2	4	13
2	5	15
3	1	20
3	2	9
3	3	18
3	5	13
4	4	15
4	5	20
5	4	15

Table qcm

idqcm	titre	Datecreation
1	Base de données	2021-09-20
2	POO	2022-04-08
3	Arbre Binaire	2022-01-09
4	Arbre Parcours	2022-02-15
5	Piles-Files	2021-12-05

1.

- a. Que retourne la requête suivante ?

```
SELECT titre FROM `qcm` WHERE date>'2022-01-10' ;
```

- b. Ecrire une requête qui donne les notes de l'élève qui a pour identifiant 4.

2.

- a. Sachant que la clé primaire de la relation `lien_eleve_qcm` est composée de l'association des deux attributs `ideleve` et `idqcm`, expliquer pourquoi avec ce schéma relationnel, un élève ne peut pas faire deux fois le même QCM.

- b. L'élève *Marty Mael* vient de faire le QCM sur la POO et a obtenu une note de 18.

Comment est/sont modifiée(s) le(s) table(s) ? Il n'est pas demandé d'écrire une requête SQL.

- c. Un nouvel élève (nom : *Lefèvre*, prenom : *Kevin*) est enregistré. Ecrire la requête permettant la mise à jour du/des relation(s).

- d. L'élève *Dubois Thomas* quitte l'établissement et toutes les références à cet élève doivent être supprimées des relations. Pour la relation `lien_eleve_qcm`, écrire la requête pour supprimer toutes les références à l'élève *Dubois Thomas* qui a pour identifiant 2.

3.

- a. Recopier et compléter les de la requête suivante pour qu'elle affiche la liste des noms et prénoms des élèves ayant fait le QCM d'idqcm égal à 4.

```
SELECT ..... FROM eleves
JOIN lien_eleve_qcm ON eleves.ideleve = .....
WHERE ..... ;
```

- b. Donner le résultat de la requête de la question a.

4. Ecrire une requête qui affiche le nom, le prénom et la note des élèves ayant fait le QCM Arbre Binaire. L'utilisation des trois tables dans la requête est attendue.

EXERCICE 4 (4 points)

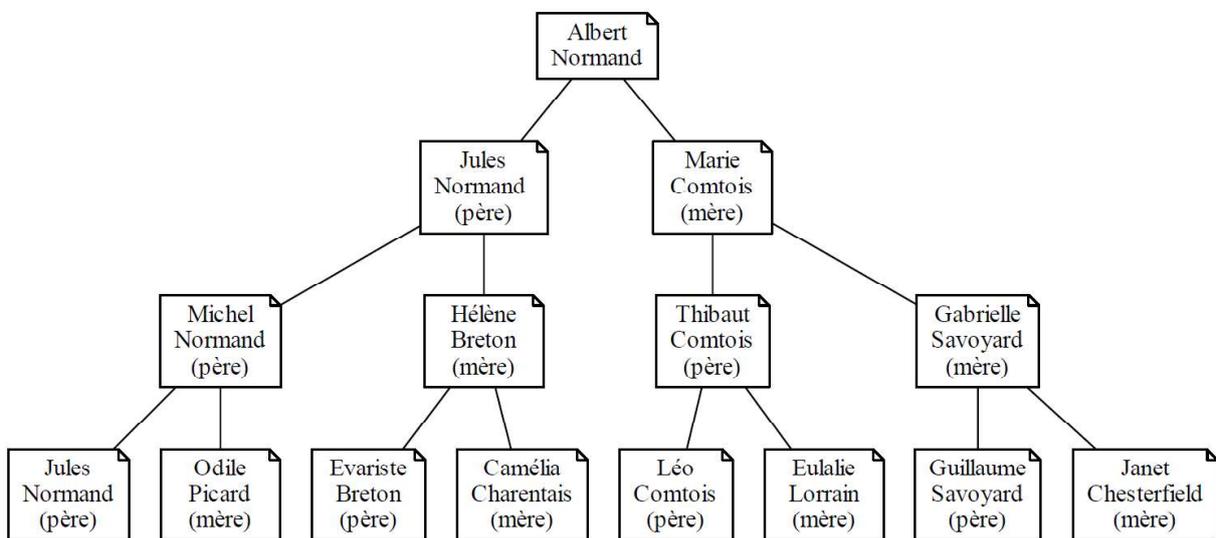
Cet exercice porte sur l'algorithmique (arbres binaires en profondeurs préfixe et infixe).

On s'intéresse dans cet exercice à l'étude d'un arbre généalogique.

Voici un extrait de l'arbre généalogique fictif d'une personne nommée Albert Normand.

L'arbre généalogique est présenté avec les parents vers le bas et les enfants vers le haut.

Albert Normand est considéré comme la génération 0. On considère ses parents comme la génération 1, ses grands-parents comme la génération 2 et ainsi de suite pour les générations précédentes.



MODELISATION DE L'ARBRE

L'arbre généalogique d'un individu est modélisé par un arbre :

- chaque nœud de l'arbre représente un individu ;
- le premier nœud du sous-arbre gauche d'un individu est associé à son père ;
- le premier nœud du sous-arbre droit est associé à sa mère.

IMPLEMENTATION DE L'ARBRE

Pour implémenter l'arbre, on utilise des notions de programmation orientée objet.

Chaque nœud de l'arbre est représenté par un **objet** qui est l'instance d'une **classe**

`Noeud` ayant trois attributs. Ainsi l'objet `N` de type `Noeud` aura les attributs suivants :

- `N.identite` de type tuple : `(prenom,nom)` de l'individu référencé par l'objet `N` ;
- `N.gauche` de type arbre binaire : le sous-arbre gauche de l'objet `N` ;
- `N.droit` de type arbre binaire : le sous-arbre droit de l'objet `N`.

Pour un individu, référencé par l'objet `N` de type `Noeud`, dont on ne connaît pas les parents, on considèrera que `N.gauche` et `N.droit` ont la valeur `None`.

1.
 - a. Expliquer en quoi cet arbre généalogique est un arbre binaire.
 - b. Pourquoi un arbre généalogique n'est pas un arbre binaire de recherche (ABR) ?
2. On souhaite obtenir la liste de tous les ascendants (ancêtres) d'Albert Normand. Pour cela, on utilise un parcours en profondeur de l'arbre.
 - a. Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur préfixe.
 - b. Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur infixe.

On donne ci-dessous le code incomplet de la fonction d'un parcours en profondeur de l'arbre, dans lequel il manque la ligne correspondant à l'instruction d'affichage du prénom et du nom de l'individu :

```
def parcours(racine_de_l_arbre) :  
    if racine_de_l_arbre != None :  
        noeud_actuel = racine_de_l_arbre  
        parcours(noeud_actuel.gauche)  
        parcours(noeud_actuel.droite)
```

- c. Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur préfixe.
 - d. Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur infixe.
3. On souhaite maintenant préciser la génération d'un individu dans l'implémentation de l'arbre généalogique. Lors de la création de l'instance, on donnera la valeur 0 par défaut.
 - a. Recopier et compléter la définition de la classe `Noeud` pour ajouter un attribut `generation` qui indique la génération d'un individu.

```
class Noeud() :  
    def __init__(prenom, nom) :  
        self.identite = (prenom, nom)  
        self.gauche = None  
        self.droite = None  
        .....
```

- b.** Ecrire la fonction récursive `numerotation` qui parcourt l'arbre et modifie l'attribut `generation` de tous les ancêtres d'Albert Normand, de sorte que les parents d'Albert Normand soient la génération 1 etc...

Cette fonction prend en paramètres `racine_de_l_arbre` de type `Noeud` et `num_gen` de type entier.

```
def numerotation(racine_de_l_arbre, num_gen=0) :  
    ...
```

- 4.** On donne la fonction suivante qui prend en paramètres l'objet `N` de type `Noeud` et la variable `affiche` de type booléen :

```
def mystere(N, affiche) :  
    if N != None :  
        if affiche :  
            print( N.identite[0])  
            mystere(N.gauche, False)  
            mystere(N.droite, True)
```

Ecrire, dans l'ordre d'affichage, le résultat de l'exécution de `mystere(racine_de_l_arbre, False)` où `racine_de_l_arbre` est le nœud qui référence Albert Normand.

ANNEXE 1 – LANGAGE SQL

- **Types de données**

CHAR(t) VARCHAR(t) TEXT	Texte fixe de t caractères. Texte de t caractères variables. Texte de 65 535 caractères max.
INT	<i>Nombre entier de -2^{31} à $2^{31}-1$ (signé) ou de 0 à $2^{32}-1$ (non signé)</i>
FLOAT	Réel à virgule flottante
DATE DATETIME	Date format AAAA-MM-JJ Date et heure format AAAA-MM-JJHH:MI:SS

- **Quelques exemples de syntaxe SQL :**

- Insérer des enregistrements :

```
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1 , valeur2)
```

- Modifier des enregistrements :

```
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur
```

- Supprimer des enregistrements :

```
DELETE FROM Table WHERE Selecteur
```

- Sélectionner des enregistrements :

```
SELECT attributs FROM Table WHERE Selecteur
```

- Effectuer une jointure :

```
SELECT attributs FROM TableA JOIN TableB ON TableA.cle1=TableB.cle2 WHERE  
Selecteur
```