BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

BAC BLANC FÉVRIER 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé. Tous documents interdits.

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

Ce sujet comporte 10 pages numérotées de 1/10 à 10/10.

Le candidat traite les 3 exercices proposés.

EXERCICE 1

Thème abordé : Bases de données (exercice librement inspiré de 22-NSIJ1JA1)

Un célèbre restaurant Ajaccien a décidé d'utiliser un SGBD (Système de Gestion de Bases de Données) pour gérer ses réservations et sa facturation. Voici le schéma relationnel de la BDD tel que l'a imaginé le concepteur :

```
Client(num_client, nom, prenom, date_naiss, email, tel)
Table(num_table, nb_couvert_table, type_table)
Facture(id_facture, montant, #num_client)
Reservation(#num_table, #num_client, date_resa, nb_pers)
```

Les chaînes de caractères dans cette base n'excèdent pas 50 caractères. On supposera que `num_client` et `id_facture` sont des entiers incrémentés automatiquement, et que chaque `num_table` est unique. On pourra s'inspirer de l'Annexe 1 pour répondre aux questions suivantes.

1. Étude du schéma relationnel
    a. Donner la clef primaire de chacune des tables : Client, Table, Facture
    b. Préciser quel est le rôle le rôle de `#num_client` dans la table Facture
    c. Quelle est la particularité de la clef primaire de la table Reservation ? Expliquer
    d. Proposer pour chaque attribut des tables Facture et Reservation son domaine (type)

2. Du MLD vers le MCD
    a. On se propose de raisonner « à l'envers », et de retrouver le MCD (Modèle Conceptuel de Données) à partir du schéma relationnel (MLD). Dessinez sur votre feuille le MCD orignal de cette BDD en prenant soin de bien y faire figurer les clefs primaires, les cardinalités, ainsi que les noms des relations
    b. Quelle relation est de type « one-to-many » (ou « many-to-one ») ?
    c. Quelle relation est de type « many-to-many » ?

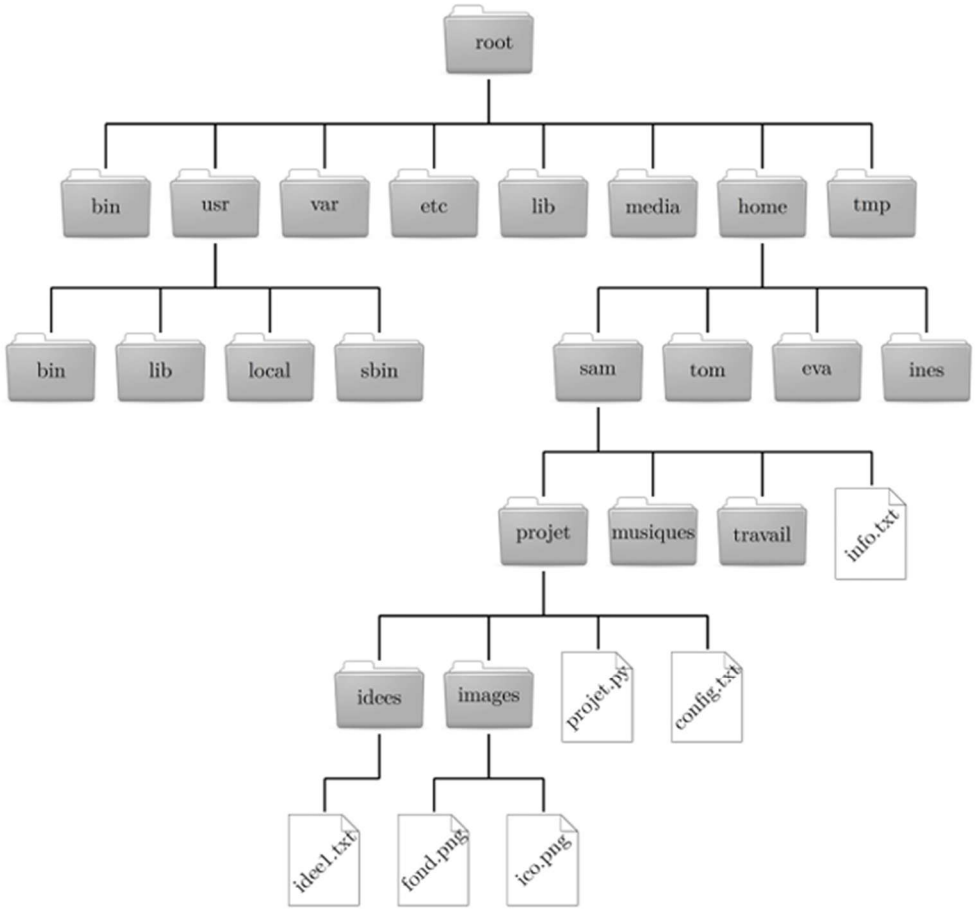
3. Recherche dans la BDD
    a. Écrire la requête permettant d'afficher tous les noms des clients dont le prénom est « Petru »
    b. Écrire la requête permettant d'afficher tous les numéros de tables de plus de 6 couverts
    c. Écrire la requête permettant d'afficher tous les identifiants des factures émises pour le client n°5 ?

4. Réflexion
    a. Peut-on avoir 2 réservations pour le client n°4 à la table n°2 à des dates différentes ? Pourquoi ? Justifier en quelques lignes.
    b. Que faudrait-il modifier dans le MCD pour répondre à cette problématique ?

EXERCICE 2

Thème abordé : système d'exploitation (22-NSIJ1JA1)

Nous avons l'arborescence ci-dessous sous un environnement Linux.



Samuel a pour nom d'utilisateur `sam`. Il a ouvert un terminal et le répertoire courant est le répertoire `musiques`. Pour tout l'exercice, on pourra tirer parti de l'annexe 2 répertoriant différentes commandes du système d'exploitation.

1. Ecrivez la ou les commande(s) qui permet(tent) de se déplacer du répertoire actuel `musiques` au répertoire `projet` :
   a. en utilisant un chemin relatif.
   b. en utilisant un chemin absolu.

2. Le répertoire courant est à présent le répertoire `sam`
   a. Ecrire la commande qui permet de lister le contenu du répertoire `projet`.
   b. Le fichier `config.txt` est protégé en écriture pour tous les utilisateurs. On souhaite modifier ce droit afin que l'utilisateur `sam` et lui seul puisse

modifier le contenu du fichier. Ecrire la commande permettant d'effectuer ce changement.

3. Le répertoire courant est toujours `sam`. L'utilisateur souhaite supprimer le répertoire `projet` en tapant l'instruction :

```
rm projet
```

Il constate que cette instruction ne fonctionne pas car ce répertoire n'est pas vide. *Finalement, il tape l'instruction :*

```
rm -R projet
```

où « R » signifie « récursif ». Le répertoire est finalement supprimé.

a. Pourquoi cette instruction fonctionne-t-elle, contrairement à la précédente ?

Les fichiers et dossiers ont été effacés dans cet ordre :

- fichier `idee1.txt`
- dossier `idees`
- fichier `fond.png`
- fichier `ico.png`
- dossier `images`
- fichier `projet.py`
- fichier `config.txt`
- dossier `projet`

b. Quel type de parcours a été réalisé par le système d'exploitation ?

4. On considère la fonction récursive suivante en langage Python :

```
def nb_fichiers(list_fich, i) :
  if i == len(list_fich) :
    return 0
  elif list_fich [i][0] == 'b' :
    return 1 + nb_fichiers(list_fich, i+1)
  else :
    return nb_fichiers(list_fich, i+1)
```

où `list_fich` est une liste contenant des noms de fichiers.

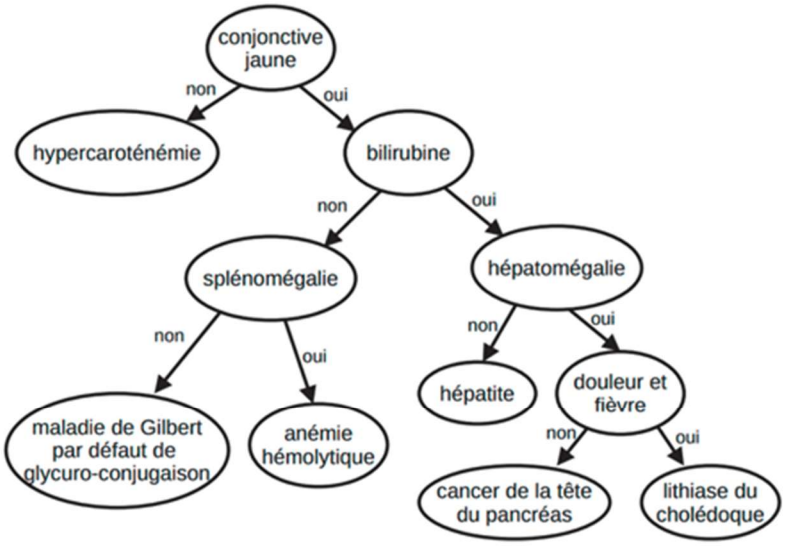
Indiquer ce que renvoie l'appel suivant en expliquant les étapes :

```
nb_fichiers(['nsi.bmp','banana.mp3','job.txt','BoyerMoore.py'],0)
```

EXERCICE 3

Thème: arbres binaires (22-NSIJ1AS1)

Les premiers travaux concernant l'aide à la décision médicale se sont développés pendant les années soixante-dix parallèlement à l'avènement de l'informatique dans le secteur médical. L'arbre de décision est une technique décisionnelle fréquemment employée pour rechercher la meilleure stratégie thérapeutique. L'arbre de décision de cet exercice, présenté ci-dessous, est un arbre binaire que l'on nommera `arb_decision`.



Arbre de décision en présence d'une jaunisse (peau anormalement jaune) chez un patient.

Rappels :

- ✓ Un **arbre binaire** est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément, appelé **nœud**, porte une étiquette.
- ✓ Le nœud initial est appelé **racine**.
- ✓ Chaque nœud d'un arbre binaire possède au plus deux **sous-arbres**.
- ✓ Chacun de ces sous-arbres est un arbre binaire, appelés sous-arbre gauche et sous-arbre droit.
- ✓ Un nœud dont les sous-arbres sont vides est appelé une **feuille**.
- ✓ Dans cet exercice, on utilisera la convention suivante : la **hauteur** d'un arbre binaire ne comportant qu'un nœud est égale à **1**.

Dans l'arbre de décision en présence d'une jaunisse chez un patient,

- ✓ un **nœud** représente un symptôme dont le médecin doit étudier la présence ou l'absence ; la réponse ne peut être que **oui** ou **non** ;
- ✓ le sous-arbre gauche d'un nœud donné décrit la démarche à adopter si le symptôme est **absent** ;
- ✓ le sous-arbre droit d'un nœud donné décrit la démarche à adopter si le symptôme est **présent** ;
- ✓ l'étiquette d'une feuille est la maladie induite par le chemin parcouru.

1. Déterminer la taille et la hauteur de l'arbre donné en exemple en introduction (arbre de décision en présence d'une jaunisse).

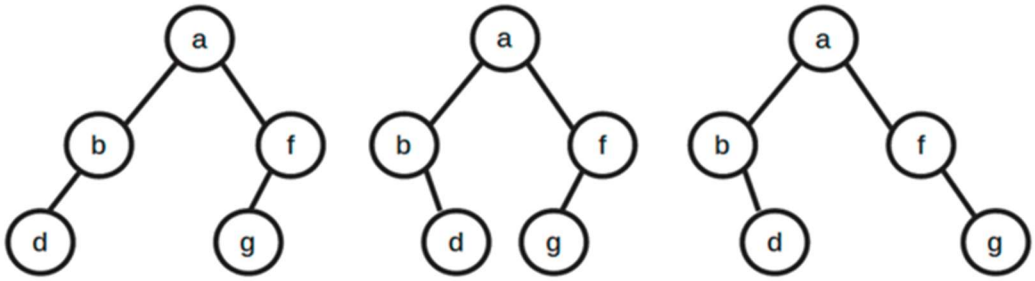
2. On choisit d'implémenter un arbre binaire à l'aide d'un dictionnaire.

```
arbre_vide = {}
arbre = {'etiquette': 'valeur' ,
         'sag': sous_arbre_gauche ,
         'sad': sous_arbre_droit }
```

Le code ci-dessous représente un arbre selon le modèle précédent.

```
{'etiquette' :'a',
 'sag':{'etiquette' :'b',
        'sag':{},
        'sad' : {'etiquette' : 'd',
                 'sag' : {},
                 'sad' : {}}},
 'sad': {'etiquette' :'f',
         'sag' : {'etiquette' : 'g',
                  'sag' : {},
                  'sad' : {}},
          'sad' : {} }}
```

a. À quelle représentation graphique correspond la structure implémentée ci-dessus ?



arbre 1

arbre 2

arbre 3

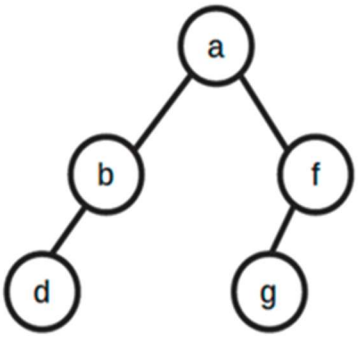
b. Représenter graphiquement l'arbre correspondant au code ci-dessous.

```
{'etiquette' :'H',
 'sag':{ 'etiquette' :'G',
        'sag': {'etiquette' : 'E',
                'sag' : {},
                'sad' : {}},
        'sad' : {'etiquette' : 'D',
                'sag' : {},
                'sad' : {'etiquette' : 'B',
                         'sag' : {},
                         'sad' : {} }}},
 'sad': {'etiquette' :'F',
         'sag' : {'etiquette' : 'C',
                  'sag' : {},
                  'sad' : {'etiquette' : 'A',
                           'sag' : {} ,
                           'sad' : {}}},
          'sad' : {} }}
```

3. La fonction `parcours(arb)` ci-dessous permet de réaliser le parcours des nœuds d'un arbre binaire `arb` donné en argument.

```
def parcours(arb):
    if arb == {}:
        return None
    parcours(arb['sag'])
    parcours(arb['sad'])
    print(arb['etiquette'])
```

a. Donner l'affichage après l'appel de la fonction `parcours` avec l'arbre dont une représentation graphique est ci-contre.



b. Écrire une fonction `parcours_maladies(arb)` qui n'affiche que les feuilles de l'arbre binaire non vide `arb` passé en argument, ce qui correspond aux maladies possiblement induites par l'arbre de décision.

4. On souhaite maintenant afficher l'ensemble des symptômes relatifs à une maladie. On considère la fonction `symptomes(arbre, mal)` avec comme argument `arbre` un arbre de décision binaire et `mal` le nom d'une maladie.
L'appel de cette fonction sur l'arbre de décision `arb_decision` de l'introduction fournit les affichages suivants.

```
>>> symptomes(arb_decision, "anémie hémolytique")
symptômes de anémie hémolytique
splénomégalie
pas de bilirubine
conjonctive jaune
```

Pour cela, on modifie la structure précédente en ajoutant une clé `surChemin` qui sera un booléen indiquant si le nœud est sur le chemin de la maladie.
La clé `surChemin` est initialisée à `False` pour tous les nœuds.

```
arbre = {'etiquette': 'valeur' ,
         'surChemin ': False ,
         'sag': 'sous-arbre gauche' ,
         'sad': 'sous-arbre droit' }
```

Recopier et compléter les lignes 6, 8, 14 et 18 du code suivant sur votre copie.

```
01    def symptomes(arb, mal):
02        if arb['sag'] != {} :
03            symptomes(arb['sag'],mal)
04
05        if arb['sad'] != {} :
06            symptomes(.....................)
07
08        if ............................. :
09            arb['surChemin'] = True
10            print('symptômes de', arb['etiquette'],':')
11
12        else :
13            if arb['sad'] != {} and arb['sad']['surChemin'] :
14                print(.....................)
15                arb['surChemin'] = True
16
17            if arb['sag'] != {} and arb['sag']['surChemin'] :
18                print(.....................)
19                arb['surChemin'] = True
```

Annexe 1

(à ne pas rendre avec la copie)

- **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	*Nombre entier de* -2^{31} à $2^{31}-1$ (signé) ou de 0 à $2^{32}-1$ (non signé)
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
```

Annexe 2

(à ne pas rendre avec la copie)

Extrait des commandes de base linux

`ls` *permet d'afficher le contenu d'un répertoire*
`cd` *se déplacer dans l'arborescence (ex* `cd repertoire1`*)*
`cp` *créer une copie d'un fichier (ex* `cp fichier1.py fichier2.py`*)*
`mv` *déplacer ou renommer un fichier ou un répertoire (*`mv fichier.txt doss`*)*
`rm` *effacer un fichier ou un répertoire (*`ex rm mon_fichier.mp3`*)*
`mkdir` *créer un répertoire (ex* `mkdir nouveau`*)*
`cat` *visualiser le contenu d'un fichier*
`chmod` *modifier les permissions d'un fichier ou d'un dossier. Pour un fichier, le format général de l'instruction est :*

```
chmod droits_user droits_group droits_other nom_fichier
```

Où `droits_user`, `droits_group` et `droits_other` indiquent respectivement les droits de l'utilisateur, du groupe et des autres et peuvent être :

```
+ ajouter
- supprimer
r read
w write
x execute
```

Exemple : `chmod rwx +r -x script.sh`