EXERCICE 3 (8 points)

Cet exercice porte sur la programmation Python (dictionnaire, récursivité, spécification), la programmation orientée objet, les bases de données relationnelles, les requêtes SQL et les arbres binaires.

Cet exercice est composé de 3 parties indépendantes.

Partie A

Dans cette partie, on s'intéresse à la gestion de la base de données d'un hôpital. On pourra utiliser les mots-clés SQL suivants : `AND`, `FROM`, `INSERT`, `INTO`, `JOIN`, `ON`, `SELECT`, `SET`, `UPDATE`, `VALUES`, `WHERE`. On utilisera également la fonction d'agrégation `COUNT` qui renvoie le nombre d'enregistrements correspondant à une requête.

La table `Patient` possède les attributs suivants :

- `nom_patient` de type TEXT (clé primaire) ;
- `prenom` de type TEXT ;
- `numero_secu` de type INT ;
- `age` de type INT.

Patient			
nom_patient	prenom	numero_secu	age
Heartman	Alice	207053523800187	17
Douglas	Bob	100017500155572	24
Woods	Caroll	258125930610747	65

La table `Symptome` possède les attributs suivants :

- `nom_patient` de type TEXT (clé primaire et clé étrangère) ;
- `toux` de type TEXT ;
- `fievre` de type TEXT ;
- `nausee` de type TEXT ;
- `anosmie` de type TEXT.

Symptome				
nom_patient	toux	fievre	nausee	anosmie
Heartman	Oui	Non	Non	Oui
Douglas	Non	Oui	Oui	Non

Symptome				
Woods	Oui	Oui	Non	Non

La table `Maladie` possède, entre autres, l'attribut `nom_maladie` de type TEXT, qui est la clé primaire. Les autres attributs de cette table ne sont pas représentés car ils ne sont pas utiles pour l'exercice.

Maladie
nom_maladie
Covid-19
Gastroentérite

La table `Diagnostic` possède les attributs suivants :

- `nom_patient` de type TEXT (clé primaire et clé étrangère) ;
- `nom_maladie` de type TEXT (clé étrangère).

Diagnostic	
nom_patient	nom_maladie
…….. …….. ……..	…….. …….. ……..

1. Écrire une requête SQL permettant d'obtenir les noms et prénoms des patients ayant strictement plus de 60 ans.
2. Alice Heartman ne tousse plus. Écrire une requête SQL permettant de mettre à jour la base de données avec cette information.
3. On souhaite effectuer des statistiques sur les symptômes des patients atteints de Covid-19. Écrire une requête SQL permettant de connaître le nombre de patients avec un diagnostic de Covid-19 qui toussent.

Un employé de l'hôpital saisit la requête suivante :

```
INSERT INTO Patients VALUES ('Douglas', 'Patrick', 168077230253829, 55)
```

4. Expliquer pourquoi cette requête produit une erreur.
5. Proposer une modification du schéma relationnel qui permettrait de résoudre ce problème.

Partie B

On s'intéresse maintenant à l'automatisation du diagnostic à partir des symptômes. Cette automatisation se fait à l'aide d'un arbre de décision binaire, tel que celui illustré sur la figure 1.

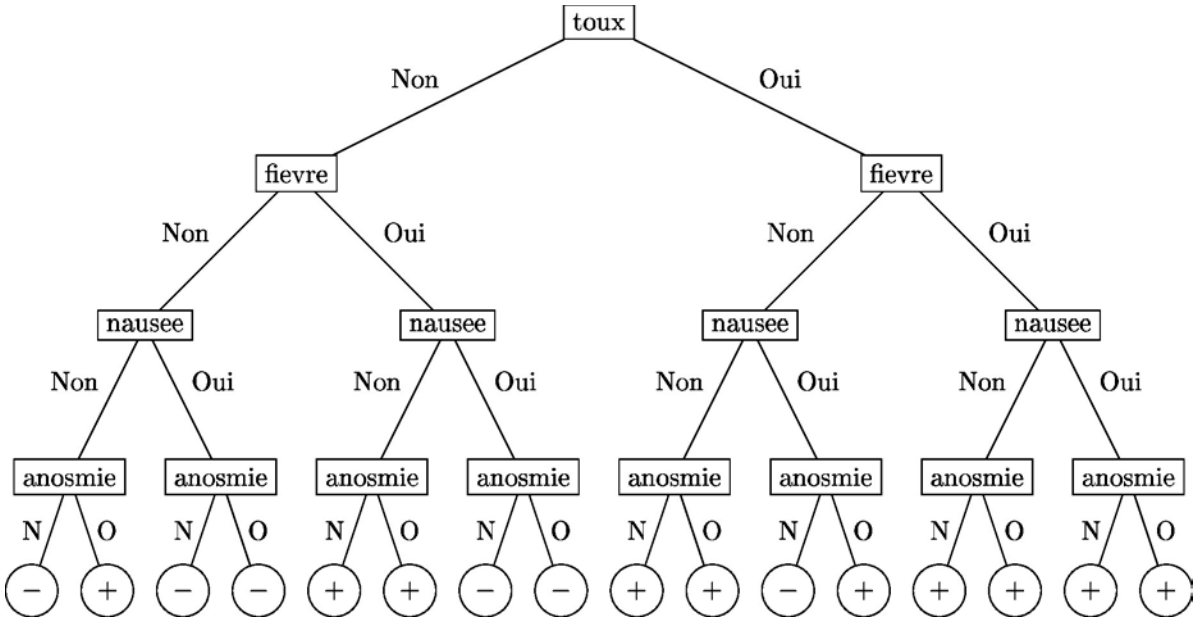

Figure 1. Exemple d'arbre de décision binaire

Chaque nœud interne de l'arbre est étiqueté par un symptôme, et chaque feuille est étiquetée par un diagnostic. Pour établir un diagnostic, on se place à la racine et on parcourt l'arbre de la manière suivante :

- si on arrive à une feuille, le diagnostic est l'étiquette de cette feuille ;
- sinon, on regarde si l'étiquette du nœud est un des symptômes du patient. Si oui, on continue le parcours dans le sous-arbre droit, sinon, on continue le parcours dans le sous-arbre gauche.

L'arbre de la figure 1 donne un diagnostic pour la Covid-19. Par exemple, un patient qui ne tousse pas et n'a pas d'anosmie, mais a de la fièvre et des nausées est diagnostiqué négatif si on suit cet arbre de décision.

6. Donner le diagnostic pour un patient qui tousse et qui a de la fièvre, mais n'a pas de nausée ni d'anosmie, d'après l'arbre de la figure 1.

On décide d'implémenter les arbres binaires à l'aide de la classe `Noeud` ci-dessous :

```
1   class Noeud:
2       def __init__(self, valeur, gauche = None, droit = None):
3           """valeur correspond au symptome si le noeud est
4           interne ou au diagnostic si le noeud est une feuille"""
5           self.valeur = valeur
6           self.gauche = gauche
7           self.droit = droit
```

```
8
9       def est_feuille(self):
10          """renvoie vrai si le noeud est une feuille faux sinon"""
11          return self.gauche == None and self.droit == None
12
13      def symptome(self):
14          assert not self.est_feuille()
15          return self.valeur
16
17      def diagnostic(self):
18          assert self.est_feuille()
19          return self.valeur
```

7. Préciser la signification de l'assertion de la méthode `symptome`.

8. Nommer un attribut et une méthode de la classe `Noeud`.

On représente les symptômes d'un patient en Python par un dictionnaire dont les clés sont les symptômes possibles, et les valeurs sont `True` si le patient présente ce symptôme et `False` sinon.
Par exemple, les symptômes du patient de la question 7 sont représentés par le dictionnaire suivant :

```
patient = {'toux' : True, 'fievre' : True, 'nausee' : False,
           'anosmie' : False}
```

9. Compléter la fonction `applique` suivante, définie récursivement, qui renvoie le diagnostic établi en utilisant un arbre de décision binaire implémenté à l'aide de la classe `Noeud` précédente.

```
def applique(arbre, patient):
    if arbre.est_feuille():
        ...
    else:
        if patient[arbre.symptome()]:
            ...
        else:
            ...
```

10. Donner la taille de l'arbre représenté en figure 1. On considère que la taille d'un arbre constitué d'une unique feuille est 1.

On souhaite réduire la taille de cet arbre en utilisant l'observation suivante : un nœud dont les deux sous-arbres sont des feuilles correspondant au même diagnostic peut être remplacé par une feuille correspondant à ce diagnostic, comme illustré en figure 2.

Figure 2. Règle de réduction pour les arbres de décision binaire

11. Appliquer cette règle à l'arbre de la figure 1 pour le réduire et dessiner le nouvel arbre.

12. Compléter la méthode `reduire` qui permet d'appliquer cette règle récursivement pour réduire la taille d'un arbre de décision binaire.

```
1  def reduire(self):
2      """fonction récursive qui réduit la taille d'un arbre de
3      décision sans changer les décisions prises"""
4      if self.est_feuille():
5           return
6      self.gauche.reduire()
7      self. ...
8      if self.gauche.est_feuille() and ... \
9         and ... == ... :
10        self.valeur = ...
11        self.gauche = ...
12        self.droite = ...
```

Partie C

Dans cette partie, on s'intéresse à l'intégrité et à la sécurité des données.

Sur les 15 chiffres du numéro de sécurité sociale, 2 servent à détecter les erreurs : étant donné le nombre n formé des 13 premiers chiffres, le nombre k formé des 2 derniers chiffres, appelé la clé, est choisi pour que n+k soit un multiple de 97.

Par exemple, 207053523800187 est bien formé car :

2070535238001 + 87 = 97 x 21345724104.

On rappelle que les opérateurs `%` et `//` permettent en Python d'obtenir respectivement le reste et le quotient dans une division euclidienne. Par exemple : `13 % 3` renvoie `1` et `13//3` renvoie `4` (car $13 = 3 \times 4 + 1$). On peut donc vérifier qu'un nombre entier n est un multiple de p en testant si le reste de la division de n par p vaut zéro.

13. Recopier et compléter la fonction `verifie` suivante qui renvoie un booléen indiquant si un numéro de sécurité sociale représenté par un entier (type `int`) est bien formé.

```
1 def verifie(num_secu):
2     n = num_secu // 100
```

```
3     k = num_secu % 100
4     return ...
```

14. Compléter la fonction `cle` qui permet de renvoyer la clé `k` d'un numéro de sécurité sociale en prenant pour paramètre le nombre `n` formé des 13 premiers chiffres du numéro de sécurité sociale.

```
1 def cle(n):
2      ...
```