CONSIGNES GÉNÉRALES

Document non autorisés
L'usage des calculatrices est interdit

Ce cahier d'examen comporte 18 pages + 2 pages supplémentaires
Les réponses doivent être écrites dans
Les espaces de réponses dédiés
En cas de besoin utiliser les pages vides en fin du cahier en le signalant dans l'espace de réponse correspondant

Il est impératif de respecter la nomenclature des objets de l'énoncé
Utiliser éventuellement l'annexe

Les énoncés et les modélisations présentés dans les deux parties ont été adaptés spécifiquement pour les besoins de l'épreuve pour des fins d'apprentissage et d'évaluation uniquement.

Le sujet comporte deux parties qui traitent les aspects suivants :

Partie I : Programmation procédurale (**Q1**..**Q8**).

Partie II : Base de données relationnelle (**Q9**..**Q25**).

Partie I : Programmation Procédurale

Le plongement lexical ou plongement sémantique (« word embedding » en anglais) est une méthode d'apprentissage d'une représentation de mots sous forme de vecteurs (représentés par des listes), utilisée notamment en traitement automatique des langages.

Afin de préparer les données textuelles pour le plongement sémantique, un ensemble d'étapes de préparation dit prétraitement est nécessaire. Une fois le prétraitement est effectué, les données peuvent être utilisées pour former un modèle de plongement lexical. Ces modèles sont souvent construits en utilisant des techniques d'apprentissage profond, telles que les réseaux de neurones. Les réseaux de neurones apprennent à représenter chaque mot par un vecteur. L'apprentissage du vecteur associé à un mot se base sur le contexte du mot extrait à partir d'un corpus de texte. Dans ce qui suit on se concentre uniquement sur les étapes de prétraitement.

Etapes du prétraitement

1. **Tokenisation :** Diviser le texte en mots ou en sous-unités plus petites, telles que les phrases ou les caractères. Cela permet de travailler sur des unités discrètes pour l'analyse.
2. **Nettoyage des données :** Éliminer les éléments indésirables tels que la ponctuation, les chiffres, les symboles, etc., qui ne sont pas pertinents pour la tâche en cours.
3. **Normalisation :** Uniformiser les mots en les convertissant en minuscules pour réduire la diversité des formes des mots.
4. **Suppression des mots vides :** Éliminer les mots courants qui n'apportent pas beaucoup de sens à la phrase, comme les articles, les pronoms, etc.
5. **Construction du vocabulaire :** Créer un vocabulaire en attribuant un index unique à chaque mot. Cela permet de convertir les mots en listes numériques.
6. **Création de paires mot-contexte :** Pour chaque mot dans le corpus, une fenêtre contextuelle est définie autour de ce mot. Pour chaque mot dans le corpus, des paires mot-contexte sont créées en associant le mot avec les mots situés à l'intérieur de sa fenêtre contextuelle.
7. **Formatage des données :** Les paires (mot de contexte, mot cible) sont préparées pour l'entraînement du modèle. Chaque paire est représentée par une liste unique où toutes les valeurs sont à zéro, sauf aux positions correspondant aux indices des deux mots de la paire, qui sont mises à 1.

Pour toute implémentation considérer les variables utilitaires suivantes :

```python
filtre = '0123456789!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\'\t\n'
```

Cette chaîne contient les caractères de ponctuation, les chiffres et les symboles.

```python
mots_vides = ['le', 'la', 'les', 'de', 'du', 'des', 'un', 'une', 'et', 'en',
              'dans', 'pour', 'par', 'sur', 'avec', 'que', 'qui', 'ce', 'il',
              'elle', 'ils', 'elles', 'tu', 'nous', 'vous', 'je']
```

Cette liste contient les mots vides (stop words) à supprimer pour améliorer la qualité du texte prétraité.

Q1. Supposons que les textes à prétraiter sont stockés dans plusieurs fichiers texte. Écrire une fonction nommée `load_corpus` qui prend en entrée une liste de chaînes de caractères `lnomF` contenant les noms des fichiers de corpus et qui retourne la liste `corpus` contenant les textes lus à partir des fichiers.

Espace de réponse pour Q1

```
def load_corpus(Lnomf):
```

Q2. Écrire la fonction `filtrage` qui prend en paramètre une chaine de caractères `text` et la chaine de caractère `filtre` (voir variables utilitaires) et permet de retourner une nouvelle chaine filtrée (pour laquelle nous avons supprimé la ponctuation, les chiffres, les symboles, etc.)

Espace de réponse pour Q2

```
def filtrage(text,filtre):
```

Q3. Écrire une fonction `tokenization` qui :

- Prend en entrée :
  - une liste de textes (`corpus`) issue de la fonction `load_corpus`,
  - une chaîne de caractères (`filtre`),
  - et une liste de mots vides (`mots_vides`).
- Retourne une liste de listes de mots, après avoir fait les traitements suivants sur chaque texte du corpus :
  1. Convertir le texte en minuscules,

2. Supprimer du texte la ponctuation, les chiffres et les symboles,
3. Diviser le texte en mots,
4. Insérer dans une liste fille de la liste finale les mots obtenus à l'étape précédente en excluant les mots vides.

Illustration sur exemple

```
Corpus=["Le chat dort paisiblement sur le tapis.",
        "La pluie tombe doucement sur le toit de la maison."]
```

Le résultat retourné est :

```
[["chat", "dort", "paisiblement", "tapis"],
 ["pluie", "tombe", "doucement", "toit", "maison"]]
```

Espace de réponse pour Q3

```
def tokenization(corpus, filtre, mots_vides):
```

Q4. Écrire la fonction `fit_on_texts` qui prend en paramètre la liste de textes `corpus`, la chaine `filtre` et la liste `mots_vides` et retourne le dictionnaire de vocabulaire `vocab`.

La structure du dictionnaire `vocab` est la suivante :

— **La clé :** un mots unique présent dans `corpus`.
— **La valeur :** est l'index unique du mot (un entier commençant par 1).

Voici un exemple illustratif avec un petit texte :

Illustration sur exemple

```
"Le chat dort paisiblement sur le tapis."
```

Le résultat retourné est :

```
vocab = { "chat" : 1, "dort" : 2, "paisiblement" : 3, "tapis" : 4}
```

Espace de réponse pour Q4

```
def fit_on_texts(corpus, filtre, mots_vides) :
```

Q5. Écrire une fonction `texts_to_sequences` qui prend en paramètre :

- une liste de textes (`corpus`),
- un dictionnaire (`vocab`),
- une chaîne de caractères (`filtre`),
- et une liste de mots vides (`mots_vides`).

Cette fonction retourne une liste de listes d'entiers(`list_index`), où chaque entier représente la valeur d'un mot dans `vocab` et ceci pour chaque mot du `corpus` tokenisé.

Illustration sur exemple

```
Corpus=["Le chat dort paisiblement sur le tapis.",
        "La pluie tombe doucement sur le toit de la maison."]
Vocab = { "chat" : 1, "dort" : 2, "paisiblement" : 3,
          "tapis" : 4, "pluie" : 5, "tombe" : 6,
          "doucement" : 7, "toit" :8, "maison" : 9}
List_index =[[1,2,3,4],[5,6,7,8,9]]
```

Espace de réponse pour Q5

```
def texts_to_sequences(corpus, filtre, mots_vides, vocab):
```

Q6. Écrire une fonction `context_pairs` qui prend en paramètre :

- une liste de mots `text` représentant un texte,
- une taille de fenêtre de contexte `window_size`(ayant 2 comme valeur par défaut).

Cette fonction retourne `pairs` une liste de tuples. Chaque tuple de `pairs` est de la forme (`mot_cible`, `mot_voisin`), où `mot_cible` est tout mot de la liste `text` et `mot_voisin` est un mot situé dans la fenêtre de contexte autour du `mot_cible`.
La figue 1 illustre un exemple de fenêtre contextuelle d'un token (`window_size = 4`).

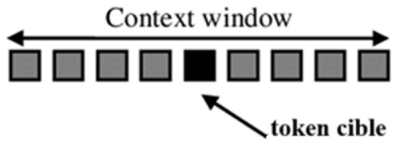


FIGURE 1 – Illustration de fenêtre contextuelle.

Illustration sur exemple

```
text =["pluie" , "tombe" , "doucement" , "toit" , "maison" ]
```

En considérant `window_size = 2` , pour le mot-cible `"doucement"` ses mots-voisins sont `"pluie"` et `"tombe"` (les deux mots qui le précèdent) et `"toit"` et `"maison"` (les deux mots qui le suivent). La liste complète des tuples générée par la fonction `context_pairs` est comme suit :

```
[('pluie', 'tombe'), ('pluie', 'doucement'), ('tombe', 'pluie'),
 ('tombe', 'doucement'), ('tombe', 'toit'), ('doucement', 'pluie'),
 ('doucement', 'tombe'), ('doucement', 'toit'), ('doucement', 'maison'),
 ('toit', 'tombe'), ('toit', 'doucement'), ('toit', 'maison'),
 ('maison', 'doucement'), ('maison', 'toit')]
```

Espace de réponse pour Q6

```
def context_pairs(text, window_size=2):
```

Q7. Écrire une fonction `one_hot_encode` qui prend en entrée :

— un tuple `pair` contenant deux mots : (`mot_cible`, `mot_voisin`)

— un dictionnaire `vocab` représentant le vocabulaire.

La fonction retourne une liste de taille égale à celle de `vocab`, initialement avec des éléments nuls, ensuite les éléments d'indices correspondants à `mot_cible` et `mot_voisin` dans `vocab` prennent la valeur 1.

Illustration sur exemple

```
pair = ("chat", "dort")
vocab = { "chat": 1, "dort": 2, "paisiblement": 3, "tapis": 4, "pluie": 5,
"tombe": 6, "doucement": 7, "toit": 8, "maison": 9 }
one_hot_encode(pair, vocab) # donne
[0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
```

Espace de réponse pour Q7

```
def one_hot_encode(pair, vocab):
```

Q8. Écrire un script Python qui :

1. Charge le contenu du fichier `"corpus.txt"` dans la liste `corpus`.
2. Construit le vocabulaire `vocab` à partir de `corpus`.
3. Génère des paires (`mot_cible`, `mot_voisin`) en utilisant une taille de fenêtre (`window_size`), où `window_size` est entrée au clavier.
4. Encode ces paires sous forme de listes one-hot.
5. Écrit le contenu des listes encodées dans un fichier `"encoded.txt"`.

Exemple :

1. Contenu du fichier `"corpus.txt"` :

```
Le chat dort paisiblement sur le tapis.
La pluie tombe doucement sur le toit de la maison.
```

2. Vocabulaire fourni :

```
vocab = { "chat": 1, "dort": 2, "paisiblement": 3,
          "tapis": 4, "pluie": 5, "tombe": 6,
          "doucement": 7, "toit": 8, "maison": 9 }
```

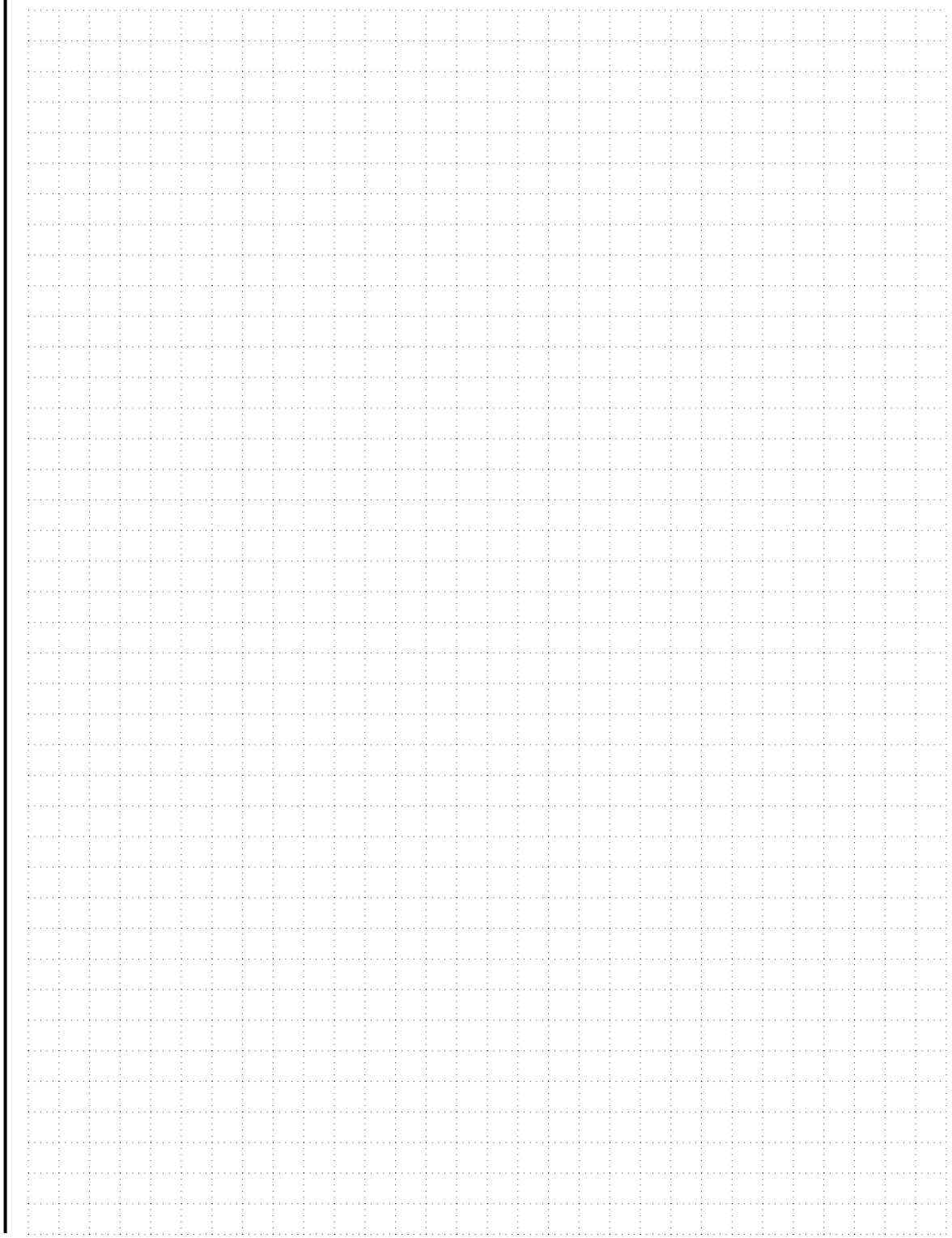
3. Taille de la fenêtre (`window_size`) :

```
window_size = 2
```

4. Résultats dans `"encoded.txt"` : Chaque ligne correspond à l'encodage one-hot d'une paire :

```
[1, 1, 0, 0, 0, 0, 0, 0, 0]  # ("chat", "dort")
[1, 0, 1, 0, 0, 0, 0, 0, 0]  # ("chat", "paisiblement")
[0, 1, 1, 0, 0, 0, 0, 0, 0]  # ("dort", "paisiblement")
[0, 1, 0, 1, 0, 0, 0, 0, 0]  # ("dort", "tapis")
[0, 0, 1, 1, 0, 0, 0, 0, 0]  # ("paisiblement", "tapis")
...
```

Espace de réponse pour Q8



Partie II : Base de Données Relationnelle

Le diagnostic du cancer, sous toutes ses formes, s'appuie sur un large éventail de techniques d'imagerie médicale avancées, conçues pour détecter avec précision d'éventuelles anomalies tissulaires. Parmi ces outils essentiels, on retrouve la mammographie, l'échographie, l'IRM, la tomodensitométrie (TDM) et la radiographie, chacune offrant des perspectives spécifiques selon la nature et la localisation de la lésion suspectée. Ces examens sont souvent complétés par une biopsie guidée par imagerie afin de confirmer la présence de cellules cancéreuses. Grâce aux progrès constants dans le domaine de l'imagerie médicale, les médecins disposent aujourd'hui de moyens performants pour affiner leurs diagnostics et adapter les traitements aux besoins spécifiques de chaque patient.

Le schéma de la base de données relationnelle qui suit fait partie du système d'information qui aide un médecin radiologue ou oncologue ou pathologiste à dépister un cancer, anticiper son traitement et épargner sa lourde prise en charge.

Le schéma relationnel de la base de données décrit ci-dessous a été élaboré pour les besoins de l'énoncé.

Base de données Carcino-.

- `Patients`(<u>`idP`</u>, `NomP`, `DnaissP`, `SexeP`, `VilleP`, `MetierP`)
  La table `Patients` enregistre les informations relatives aux patients qui viennent consulter par des techniques d'imagerie.
  - `idP` : identifiant d'un patient de type entier clé primaire.
  - `NomP` : nom du patient, de type chaîne de caractères.
  - `DnaissP` : date de naissance du patient, de type date selon le format `"AAAA-MM-JJ"`.
  - `SexeP` : sexe du patient, de type caractère (`"F"` : Féminin, `"M"` : Masculin).
  - `VilleP` : ville du patient, de type chaîne de caractères.
  - `MetierP` : métier du patient, de type chaîne de caractères.
- `Medecins`(<u>`idM`</u> , `NomM`, `SpecM`, `EmailM`, `TelM`)
  La table `Medecins` enregistre les informations relatives aux médecins qui traitent les patients.
  - `idM` : identifiant d'un médecin, de type entier clé primaire.
  - `NomM` : nom du médecin, de type chaîne de caractères.
  - `SpecM` : spécialité du médecin, de type chaîne de caractères (ex. : `"Radiologue"`, `"Oncologue"`).
  - `EmailM` : adresse e-mail du médecin, de type chaîne de caractères.
  - `TelM` : numéro de téléphone du médecin, de type chaîne
- `Imageries`(<u>`idI`</u>, `TypeI`, `DateI`, #`idP`, #`idM`)
  La table `Imageries` enregistre les informations relatives aux examens d'imagerie réalisés pour chaque patient.
  - `idI` : identifiant d'une imagerie de type entier clé primaire.
  - `TypeI` : type d'imagerie réalisée (ex. : Mammographie, Échographie, IRM (Imagerie par Résonance Magnétique), TDM (Tomodensitométrie) et Radiographie, Biopsie à l'aiguille), de type chaîne de caractères.
  - `DateI` : date de l'examen d'imagerie, de type date selon le format `"AAAA-MM-JJ"`.

— `idP` : identifiant du patient concerné, de type entier clé étrangère fait référence à la table Patients.

— `idM` : identifiant du médecin ayant ordonné l'imagerie, de type entier clé étrangère fait référence à la table Medecins.

■ `Staff`(#`idM`, #`idI`)
La table `Staff` enregistre les associations entre les médecins qui ont collaboré avec le médecin donneur d'ordre de l'imagerie sur une imagerie spécifique pour se concerter et prendre une décision de traitement.

— `idM` : identifiant d'un médecin, de type entier clé étrangère fait référence à la table Medecins.

— `idI` : identifiant d'une imagerie, de type entier clé étrangère fait référence à la table Imageries.

La combinaison des clés étrangères `idM` identifiants du médecin et `idI` de l'imagerie forment la Clé primaire de la table, garantissant l'unicité de chaque collaboration.

■ `GroupementCellulaires`(`idGC`, `X`, `Y`, `Largeur`, `Longueur`, `Profondeur`, `Couleur`, `Texture`, `Douteux`, #`idI`)
La table `GroupementCellulaires` enregistre les informations sur les groupements cellulaires observés lors des examens d'imagerie.

— `idGC` : identifiant d'un groupement cellulaire, de type entier clé primaire.

— `X` : coordonnée X du groupement, de type réel.

— `Y` : coordonnée Y du groupement, de type réel.

— `Largeur` : largeur du groupement, de type réel.

— `Longueur` : longueur du groupement, de type réel.

— `Profondeur` : profondeur du groupement, de type réel.

— `Couleur` : couleur du groupement, de type chaîne de caractères.

— `Texture` : texture du groupement, de type chaîne de caractères.

— `Douteux` : indique si le groupement est suspect, de type booléen(`True` ou `False`).

— `MasqueROI` : chemin d'accès à un fichier contenant un masque qui délimite le groupement cellulaire, de type chaîne de caractères.

— `idI` : identifiant de l'imagerie, de type entier clé étrangère fait référence à la table Imageries.

NB :

— `X`, `Y`, `Largeur`, `Longueur`, `Profondeur` sont exprimées en millimètre par rapport aux coordonnées de l'organe examiné.

— Le fichier contenu dans `MasqueROI` contient un masque couvrant la région d'intérêt marquant le groupement cellulaire. Se présente sous forme d'une matrice booléenne, $M_{yx} =$ `True` si les coordonnées (y, x) appartiennent au groupement cellulaire.

■ `Diagnostics`(`idD`, `Decision`, `PCMD`, `RapportD`, `DateD`, `Recommandation`, #`idI`)
La table `Diagnostics` enregistre les informations relatives aux diagnostics effectués sur les imageries.

— `idD` : identifiant unique du diagnostic, de type entier clé primaire.

— `Decision` : diagnostic principal global (ex. : Bénin, Malin), de type chaîne de caractères.

— `PCMD` : pourcentage de cellules malines observées dans la totalité de l'imagerie, de type réel.

— `RapportD` : le rapport élaboré par le staff ou le médecin responsable détaillant le diagnostic, de type chaîne de caractères.
— `DateD` : date à laquelle l'analyse a été réalisée, de type date selon le format `"AAAA-MM-JJ"`.
— `Recommandation` : suggestions de chirurgie ou tout autre examen complémentaire pour le patient, de type chaîne de caractères.
— `idI` : identifiant de l'imagerie associée, de type entier, clé étrangère qui fait référence à la table Imageries.

Travail demandé

Algèbre relationnelle

Dans la suite on suppose que les tables de la base de données sont remplies en respectant les contraintes et les règles de gestion su citées.

Exprimer en algèbre relationnelle les requêtes permettant de :

Q9. Lister les identifiants des patients qui ont effectué uniquement des imageries de type `"IRM"`.

Espace de réponse pour Q9

Q10. Donner les types d'imageries pour lesquelles on a identifié des groupements cellulaires douteux.

Espace de réponse pour Q10

Q11. Donner les noms des patients (sexe féminin) ayant effectué une imagerie de type `"Mammographie"` dont la décision est `"Malin"`.

Espace de réponse pour Q11

SQL

Q12. La requête algébrique suivante permet de lister les noms des médecins ayant participé à des staffs pour des imageries dont les diagnostics ont abouti à des `PCMD` supérieur à 50%. On demande de traduire la requête algébrique donnée en SQL.

$$\prod_{\texttt{NomM}} \left(\sigma_{\texttt{PCMD}>50.0} \left(\texttt{Medecins} \bowtie_{\texttt{idM}} \texttt{Staff} \bowtie_{\texttt{idI}} \texttt{Diagnostics} \right) \right)$$

Espace de réponse pour Q12

Q13. Lister les villes où on recense la présence de plus de 30 cas de femmes pour lesquelles on a abouti à des décisions d'imageries autre que (`"Bénin"` et `"Normal"`).

Espace de réponse pour Q13

Q14. Pour chaque identifiant d'imagerie calculer selon l'ordre décroissant, le total de médecins qui ont fait staff pour l'examiner. Présenter le résultat par ordre décroissant du total des médecins.

Espace de réponse pour Q14

Q15. Lister les noms et les âges des patients pour qui les imageries ont nécessité des staffs de plus de trois médecins pour dépister leurs cas.
Il est conseillé de s'inspirer de la requête suivante qui utilise une fonction prédéfinie `julianday(date)` pour calculer les âges des patients à partir de leurs dates de naissance.

```sql
SELECT  (julianday('now') - julianday(date_naissance)) / 365 AS Age
FROM Patients;
```

Espace de réponse pour Q15

Q16. Déterminez tous les détails des imageries qui pour le moment n'ont pas de diagnostic.

Espace de réponse pour Q16

Q17. Lister les id et les dates des imageries qui n'ont pas conclu à des groupements cellulaires.

Espace de réponse pour Q17

Q18. Lister les informations des patients avec des mammographies présentant un diagnostic où le pourcentage de cellules malines dépasse 20%.

Espace de réponse pour Q18

Q19. Pour chaque identifiant de médecin donner le nombre ordonné de chaque type d'imagerie (nombre de mammographie, le nombre d'IRM etc.)

Espace de réponse pour Q19

Q20. Donner les noms, les spécialités et adresses mail des médecins qui n'ont jamais ordonné d'imageries et qui n'ont jamais fait de staff pour diagnostiquer une imagerie.

Espace de réponse pour Q20

Q21. Donner la liste des médecins qui n'ont jamais ordonné d'imageries mais qui ont assisté à des staffs pour diagnostic d'imageries.

Espace de réponse pour Q21

SQLITE

L'objectif dans cette partie est de récupérer les données des diagnostics des patients depuis la base de données SQLite `"carcino-"` , de les structurer par ville et sexe, puis de générer un tableau de statistique présentant, pour chaque ville, le nombre de cas masculins et le nombre de cas féminins pour chaque type de diagnostic (`"Malin"`, `"Bénin"`, `"Normal"`), ainsi que le dénombrement des patients correspondants.

Dans la suite, les fonctions demandées doivent être écrites en Python en désignant par `cur` le curseur d'exécution des requêtes.

Q22. Écrire une fonction `villesPatients` qui prend en entrée `cur` le curseur de la connexion vers la base de données. La fonction retourne un ensemble contenant toutes les villes des patients.

Espace de réponse pour Q22

```
def villesPatients(cur):
```

Q23. Écrire une fonction `decisionsDiagnostics` qui prend en entrée `cur` le curseur de la connexion vers la base de données. La fonction retourne un ensemble contenant toutes les décisions prises pour les diagnostics des patients.

Espace de réponse pour Q23

```
def decisionsDiagnostics(cur):
```

Q24. Écrire une fonction `denombrerCas` qui prend en entrée :

- `cur` le curseur de la connexion vers la base de données.
- `ville` une chaine contenant la ville des patients.
- `sexe` un caractère contenant le sexe des patients ('F' ou 'M').
- `decision` une chaine contenant une décision de diagnostic.
- `annee` un entier contenant une année.

La fonction retourne le nombre de patients de la ville ville de sexe sexe qui ont été diagnostiqués avec la décision `decision` durant l'année `annee` (l'attribut `DateD` de la table `Diagnostics`) .

Espace de réponse pour Q24

```
def denombrerCas(cur, ville, sexe, decision, annee):
```

Q25. Écrire une fonction nommée `statsAnnee` qui prend en entrée :

- `cur` le curseur de la connexion à la base de données.
- `annee` un entier contenant une année.

La fonction doit retourner un dictionnaire où :

- chaque clé est un tuple de la forme (ville, sexe, decision).
- chaque valeur est le nombre de cas qui ont été diagnostiqués dans cette ville, pour ce sexe et pour cette décision.

Le dictionnaire doit contenir toutes les combinaisons possibles de villes, de sexes et de décisions qui existent dans la base de données.

Espace de réponse pour Q25

```
def statsAnnee(cur, annee):
```

Espace de réponse supplémentaire

Espace de réponse supplémentaire

Annexe

Quelques Fonctions/Méthodes Python/Schéma Relationnel

Sans aucune obligation, les fonctions suivantes pourraient vous être utiles.

Opérations sur les itérables (`str`, `tuple`, `list`, `dict`, `etc.`)

- `len(it)` retourne le nombre d'éléments de l'itérable it.
- `range(d,f,p)` retourne la séquence des valeurs entières successives comprises entre d et f, f exclu, par $pas = p$.
- `sum(it)` retourne la somme de tous les éléments de l'itérable it.
- `min(it)` retourne la valeur minimale de l'itérable it.
- `min(it, key = fct)` retourne la valeur minimale de l'itérable it où la fonction key définit le critère de comparaison.
- `max(it)` retourne la valeur maximale de l'itérable it.
- `max(it, key = fct)` retourne la valeur maximale de l'itérable it où la fonction key définit le critère de comparaison.
- `sum(it)` retourne la somme des éléments de l'itérable it.
- `x in it` vérifie si x appartient à it.
- `sorted(it)` retourne une liste contenant les éléments de it dans l'ordre croissant.
- `sorted(it, key = fct, reverse = True)` retourne une liste contenant les éléments de it dans l'ordre décroissant où la fonction key définit le critère de comparaison.
- `lst.sort()` trie la liste lst dans l'ordre croissant.
- `lst.sort(key = fct, reverse = True)` trie la liste lst en ordre décroissant en utilisant key comme critère de comparaison.
- `lst.extend(iterable)` permet d'ajouter tous les éléments de l'objet `iterable` (par exemple, une liste ou un tuple) à la fin de la liste `l`.
- `lst.count(val)` retourne le nombre d'occurrences de val dans la liste lst.
- `lst.append(val)` ajoute val à la fin de la liste lst.
- `lst.remove(val)` supprime la première occurrence val de la liste lst.
- `lst.pop(idx)` supprime puis retourne la valeur située à la position `idx` de la liste lst.
- `lst.index(val)` retourne l'indice de la première occurrence val de la liste lst.
- `chaine.lower()` retourne une chaîne de caractères où toutes les lettres majuscules sont converties en lettres minuscules.
- `chaine.upper()` retourne une chaîne de caractères où toutes les lettres minuscules sont converties en lettres majuscules.
- `chaine.isalphanum()` retourne `True` si la chaîne est composée uniquement de caractères alphanumériques (lettres et chiffres), et `False` sinon.
- `chaine.replace(ancien, nouveau)` retourne une chaîne de caractères où toutes les occurrences de `ancien` sont remplacées par `nouveau`.
- `chaine.find(sous_chaine)` retourne l'index de la première occurrence de `sous_chaine` dans la chaîne, ou `-1` si `sous_chaine` n'est pas trouvée.
- `source.split(motif)` retourne une liste formée par des chaînes de caractères résultantes du découpage de la chaîne source autour de la chaîne motif.
- `motif.join(itérable de chaînes)` retourne une chaîne de caractères résultante de la concaténation des éléments de l'itérable intercalés par le motif.

— `motif.format(paramètres)` retourne une chaîne de caractères obtenue en substituant dans l'ordre chaque caractère dans motif par un objet dans paramètres.
— `d.values()` retourne un itérable formé par les valeurs du dictionnaire d.
— `d.items()` retourne un itérable de couples (k,v) ou k est une clé du dictionnaire d et v est la valeur associée.

Opérations sur les fichiers :

— `f=open(nomF,m)` permet d'ouvrir le fichier `nomF` en mode `m` ou `m='r'` ou `'w'`.
— `f.close()` permet de fermer un fichier.
— `f.read()` permet de lire et retourner le contenu d'un fichier dans une chaîne de caractères.
— `f.readline()` permet de lire et retourner la ligne courante d'un fichier dans une chaîne de caractères.
— `f.readlines()` permet de lire et retourner le contenu de toutes les lignes d'un fichier dans une liste.
— `f.write(chaine)` permet d'écrire la chaîne de caractères `chaine` dans le fichier. Retourne le nombre de caractères écrits.
— `f.writelines(iterable)` permet d'écrire les éléments de l'objet `iterable` (par exemple, une liste de chaînes) dans le fichier, sans ajouter de nouvelle ligne automatiquement.

Module sqlite3

— `cur.execute(req)` exécuter la requête SQL décrite par le `str` req à partir du curseur `cur`.
— `cur.execute(req, seq)` exécuter la requête paramétrée décrite en SQL par le `str` req à partir du curseur `cur`.
— `cur.fetchall()` récupère tous les résultats de la dernière requête exécutée avec le curseur `cur` et les retourne sous forme de liste de tuples.

Schéma relationnel BD

- Patients(idP, NomP, DnaissP, SexeP, VilleP, MetierP)
- Medecins(idM , NomM, SpecM, EmailM, TelM)
- Imageries(idI, TypeI, DateI, #idP, #idM)
- Staff(#idM, #idI)
- GroupementCellulaires(idGC, X, Y, Largeur, Longueur, Profondeur, Couleur, Texture, Douteux, #idI)
- Diagnostics(idD, Decision, PCMD, RapportD, DateD, Recommandation, #idI)