



Concours Biologie et Géologie Epreuve d'Informatique + Correction

Date : Samedi 26 Juin 2021 Heure : 8 H Durée : 2 H Nombre de pages : 7

Barème : **PROBLEME 1 : 9 points**
PROBLEME 2 : 11 points

DOCUMENTS NON AUTORISES
L'USAGE DES CALCULATRICES EST INTERDIT
IL FAUT RESPECTER IMPERATIVEMENT LES NOTATIONS DE L'ENONCE
VOUS POUVEZ EVENTUELLEMENT UTILISER LES FONCTIONS PYTHON
DECRIRES A L'ANNEXE (PAGE 7)

PROBLEME 1

Dans ce problème, on s'intéresse à implémenter quelques fonctions pour manipuler les régions extraites d'une image en niveaux de gris.

Il est à noter que le problème peut être traité sans aucun prérequis sur le domaine du traitement d'images.

Dans ce qui suit nous présentons quelques notions de base puis une description de ces fonctions.

Notions de base

- Une image en niveaux de gris est une collection de pixels organisés sous forme matricielle, où chaque pixel est défini par :
 - ses coordonnées (y, x) où y est l'indice de ligne et x est l'indice de colonne ;
 - sa valeur, parmi 256 valeurs possibles entre le noir et le blanc (0 pour le noir et 255 pour le blanc), représentant le niveau de gris du pixel.
- Une image dite normalisée, associée à une image donnée en niveau de gris, est une image où les niveaux de gris sont des réels dans l'intervalle $[0,1]$.
- Une région de l'image est un ensemble de pixels.
- Une région uniforme est formée par des pixels ayant exactement le même niveau de gris (même intensité lumineuse).
- Les 8-voisins d'un pixel p de coordonnées (y, x) sont ses voisins immédiats, comme l'illustre la figure ci-après.

- Deux pixels **p** et **q**, n'appartenant pas aux bords de l'image, sont dits voisins si et seulement si **p** est l'un des 8-voisins immédiats de **q** ou réciproquement.

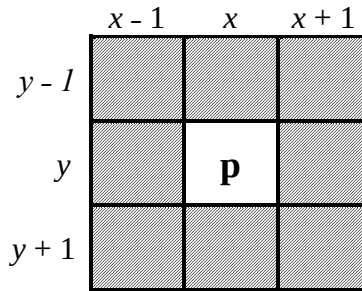


Figure : Les 8-voisins immédiats (cases hachurées) du pixel **p**.

- La frontière d'une région **r** est formée par les pixels qui possèdent un voisin extérieur à la région **r**.
- L'intérieur d'une région **r** est formé par les pixels dont tous les voisins immédiats sont dans la région **r**.
- Un masque associé à une image en niveaux de gris, est une image en noir et blanc où chaque pixel a une valeur booléenne, `False` pour le noir et `True` pour le blanc.

Hypothèses de travail

- Les réponses seront écrites en langage Python.
- Les tableaux à une ou à deux dimensions (vecteurs et matrices) sont représentés par des instances de la classe `numpy.ndarray`.
- Une région de l'image est représentée par un ensemble (instance de la classe `set`) de tuples **p** (**p** = (**y**, **x**)) décrivant les coordonnées des pixels appartenant à cette région.
- La bibliothèque `numpy` est importée par : `import numpy as np`.

Travail demandé

1. Écrire une fonction, nommée **normaliser**, qui prend en entrée une matrice **img**, calcule pixel par pixel la matrice **nimg** représentant l'image en niveaux de gris normalisée de même taille. Pour un pixel de coordonnées (**y**, **x**), la formule de normalisation est :

$$nimg_{y,x} = \frac{img_{y,x} - v_{min}}{v_{max} - v_{min}}$$

où :

- v_{min} est le niveau de gris le plus faible dans **img**.
- v_{max} est le niveau de gris le plus élevé dans **img**.
- v_{min} n'est jamais égale à v_{max} .
- $0 \leq y < h$ avec h le nombre de lignes de **img**.
- $0 \leq x < w$ avec w le nombre de colonnes de **img**.

Cette fonction retourne la matrice **nimg**.

```
def normaliser(img):
    vmin = img.min()
    vmax = img.max()
    nimg = np.empty(img.shape, dtype = np.float)
    h, w = img.shape
    for y in range(h):
        for x in range(w):
            nimg[y,x] = (img[y,x] - vmin) / (vmax - vmin)
    return nimg
#version2
def normaliser(img):
    imin = img.min()
    imax = img.max()
    return (img - imin) / (imax - imin)
```

2. Écrire une fonction, nommée **voisins**, qui prend en entrée un tuple **p** contenant les coordonnées d'un pixel et retourne un ensemble formé par les coordonnées des 8 pixels voisins immédiats du pixel de coordonnées **p**. Aucun traitement particulier n'est à prévoir pour les pixels associés aux bords de l'image.

```
def voisins(p):
    y, x = p
    res = set()
    for yp in range(y-1, y+2):
        for xp in range(x-1, x+2):
            res.add((yp, xp))
    res.remove((y, x))
    return res
#version2
def voisins(p):
    y, x = p
    res = set()
    for i in range(-1, 2):
        for j in range(-1, 2):
            res.add((y+i, x+j))
    return res - {p}
```

3. Écrire une fonction, nommée **sont_voisins**, qui prend deux tuples **p** et **q** représentant les coordonnées de deux pixels, retourne **True** si **p** et **q** sont voisins et **False** sinon.

```
def sont_voisins(p, q):
    return p in voisins(q)
```

4. Écrire une fonction, nommée **creer_masque**, qui prend en entrée :

- un ensemble de tuples, noté **r**, représentant une région ;
- un tuple de 2 entiers, noté **taille**, contenant respectivement la hauteur et la largeur du masque à retourner.

Cette fonction retourne la matrice **masque** où les pixels relatifs à la région **r** sont blancs et les restants sont noirs.

```
def creer_masque(r, taille):
    img = np.zeros(taille, np.bool)
    for y, x in r:
        img[y,x] = True
    return img
#version 2
def creer_masque(r, taille):
    res = np.zeros(shape = taille, dtype = np.bool)
    y, x = zip(*r)
    res[y,x] = True
    return res
#version 3
def creer_masque(r, taille):
    h, w = taille
    return np.array([(y,x) in r for x in range(w)] for y in range(h))
```

5. Écrire une fonction, nommée **est_uniforme**, qui prend en entrée :

- un ensemble de tuples, noté **r**, représentant une région ;
- une matrice **img** représentant une image en niveaux de gris.

Cette fonction retourne **True** si la région **r** est uniforme et **False** sinon.

```
def est_uniforme(r, img):
    s = {img[y,x] for y, x in r}
    return len(s) == 1
#version 2
def est_uniforme(r, img):
    r = list(r)
    ref = img[r[0]]
    for y, x in r[1:]:
        if img[y,x] != ref:
            return False
    return True
```

6. Écrire une fonction, nommée **frontiere**, qui prend en entrée un ensemble de tuples représentant une région **r**, retourne un ensemble **front** contenant les coordonnées des pixels appartenant à la frontière de **r**.

```
def frontiere(r):
    return {p for p in r if any(q not in r for q in voisins(p))}
#version 2
def frontiere(r):
    s = set()
    for p in r:
        for q in voisins(p):
            if q not in r:
                s.add(p)
                break
    return s
```

-
7. Écrire une fonction, nommée **interieur**, qui prend en entrée un ensemble de tuples représentant une région **r** et retourne un ensemble **inter** contenant les coordonnées des pixels appartenant à l'intérieur de **r**.

```
def interieur(r):
    return r - frontiere(r)
#version 2
def interieur(r):
    return {p for p in r if p not in frontiere(r)}
```

PROBLEME 2

Une entreprise agricole implémente une base de données relationnelle pour gérer les interventions auprès d'agriculteurs propriétaires de parcelles de terrains (portion de terrain pour usage agricole) ainsi que les paies mensuelles des intervenants.

Au besoin, un agriculteur fait appel à cette entreprise pour accomplir une ou plusieurs interventions à travers un ou plusieurs intervenants (ouvriers, techniciens, ingénieurs, etc.) spécialisés payés, selon un salaire journalier, à la fin de chaque mois.

Contraintes à considérer

- Un agriculteur possède une ou plusieurs parcelles de terrain.
- Une parcelle de terrain appartient à un et un seul propriétaire.
- L'entreprise paie ses intervenants mensuellement, en fonction du nombre de jours de leurs interventions.
- Une intervention est un travail effectué par un intervenant pendant un certain nombre de jours consécutifs sur une parcelle de terrain.
- Une intervention ne dépasse pas 28 jours.
- Une intervention peut être accomplie au cours d'un mois ou bien s'étaler sur deux mois consécutifs.

- Un intervenant est considéré comme un employé de l'entreprise.
- Un intervenant ne peut pas démarrer une nouvelle intervention avant de terminer l'intervention en cours.

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

▪ **Agriculteur** (idAg, nomAg, prenomAg, villeAg)

La table **Agriculteur** enregistre les informations concernant un agriculteur :

- idAg : identifiant de l'agriculteur (numéro de CIN) de type chaîne de caractères, clé primaire ;
- nomAg : nom de l'agriculteur de type chaîne de caractères ;
- prenomAg : prénom de l'agriculteur de type chaîne de caractères ;
- villeAg : ville de l'agriculteur de type chaîne de caractères.

▪ **Parcelle** (idPar, designation, villeP, superficie, #idAg)

La table **Parcelle** enregistre les informations des parcelles de terrain des agriculteurs :

- idPar : identifiant de la parcelle de terrain de type chaîne de caractères, clé primaire ;
- designation : nom de la parcelle de terrain de type chaîne de caractères ;
- villeP : ville où se situe la parcelle de terrain de type chaîne de caractères ;
- superficie : superficie de la parcelle de terrain exprimée en hectare de type réel ;
- idAg : identifiant de l'agriculteur propriétaire de la parcelle de terrain de type chaîne de caractères, clé étrangère qui fait référence à la table **Agriculteur**.

▪ **Intervenant** (idIn, nomIn, prenomIn, villeIn, salaireJ)

La table **Intervenant** enregistre les informations relatives aux intervenants de l'entreprise agricole :

- idIn : identifiant d'un intervenant (numéro de CIN) de type chaîne de caractères, clé primaire ;
- nomIn : nom d'un intervenant de type chaîne de caractères ;
- prenomIn : prénom d'un intervenant de type chaîne de caractères ;
- villeIn : ville d'un intervenant de type chaîne de caractères ;
- salaireJ : salaire journalier d'un intervenant de type réel.

▪ **Intervention** (#idIn, #idPar, dateDebut, nbJours)

La table **Intervention** enregistre les interventions des intervenants :

- idIn : identifiant d'un intervenant de type chaîne de caractères, clé étrangère qui fait référence à la table **Intervenant** ;
- idPar : identifiant d'une parcelle de terrain de type chaîne de caractères, clé étrangère qui fait référence à la table **Parcelle** ;
- dateDebut : date du début d'une intervention de type date au format "AAAA-MM-JJ" ;
- nbJours : durée effective de l'intervention en nombre de jours consécutifs de type entier.

Partie 1

Exprimer en algèbre relationnelle les requêtes permettant de :

1. Donner les noms, prénoms et villes des grands agriculteurs. On appelle grand agriculteur celui qui possède au moins une parcelle de terrain d'une superficie de plus de 10 hectares.

$$\Pi_{\text{nomAg, prenomAg, villeAg}} \left(\sigma_{\text{superficie} \geq 10} \left(\text{Agriculteur} \bowtie_{\text{idAg}} \text{Parcelle} \right) \right)$$

2. Donner les identifiants des intervenants qui possèdent des parcelles de terrains.

$$\Pi_{\text{idAg}} (\text{Parcelle}) \cap \Pi_{\text{idIn}} (\text{Intervenant})$$

ou

$$\Pi_{\text{idAg}} \left(\text{Parcelle} \bowtie_{\text{Parcelle.idAg} = \text{Intervenant.idIn}} \text{Intervenant} \right)$$

Partie 2

Exprimer en SQL les requêtes suivantes permettant de :

1. Créer la table **Agriculteur** en respectant les contraintes spécifiées ci-dessus.

Dans la suite on suppose que les tables de la base de données sont toutes créées et remplies en respectant les contraintes déjà énumérées.

```
CREATE TABLE Agriculteur (
    idAg TEXT,
    nomAg TEXT,
    prenomAg TEXT,
    villeAg TEXT,
    PRIMARY KEY(idAg)
);
/* ou bien */
CREATE TABLE Agriculteur (
    idAg TEXT PRIMARY KEY,
    nomAg TEXT,
    prenomAg TEXT,
    villageAG TEXT
);
```

-
2. Ajouter à la table **Agriculteur** les informations suivantes : '33333333', 'DHHLMZ', 'AJFHHL', 'Béjà'.

```
INSERT INTO Agriculteur
VALUES ("33333333", "DHHLMZ", "AJFHHL", "Beja");
```

-
3. Appliquer une augmentation tarifaire de 5% pour les salaires journaliers des intervenants qui ont fait au moins une intervention sur des parcelles de terrain de plus de 5 hectares et ce pour les gratifier.

```
UPDATE Intervenent
SET salaireJ = salaireJ * 1.05
WHERE idIn IN (
    SELECT idIn
    FROM Intervention AS It, Parcelle AS P
    WHERE It.idPar = P.idPar AND superficie >= 5
);
```

-
4. Supprimer les intervenants n'ayant effectués aucune intervention depuis 2015.

```
DELETE FROM Intervenent
WHERE idIn NOT IN (
    SELECT idIn
    FROM Intervention
    WHERE dateDebut > "2015-01-01"
);
/* ou bien */
DELETE FROM Intervenent
WHERE idIn NOT IN (
    SELECT idIn
    FROM Intervention
    WHERE dateDebut > "2015%"
);
```

-
5. Déterminer toutes les informations relatives aux intervenants triées par ordre décroissant selon le salaire journalier.

```
SELECT *
FROM Intervenent
ORDER BY salaireJ DESC;
```


-
6. Donner les noms des agriculteurs qui ont fait intervenir des intervenants portant les mêmes noms de famille qu'eux.

```
SELECT nomAg
FROM Agriculteur AS A, Parcelle AS P, Intervenant AS I,
Intervention AS It
WHERE P.idPar=It.idPar
AND I.idIn=It.idIn
AND A.idAg=P.idAg
AND A.nomAg=I.nomIn;
```

-
7. Calculer le nombre de propriétaires de parcelles par ville originaires de cette même ville.

```
SELECT COUNT(*), A.idAg
FROM Agriculteur A, Parcelle P
WHERE A.idAg = P.idAg
GROUP BY A.idAg;
```

-
8. Déterminer les noms et les prénoms des agriculteurs possédant plus que trois parcelles.

```
SELECT nomAg, prenomAg
FROM Parcelle P, Agriculteur A
WHERE P.idAg = A.idAg
GROUP BY A.idAg
HAVING COUNT(*) >= 3;
```

-
9. Déterminer les noms des intervenants ayant le salaire journalier le plus élevé.

```
SELECT nomIn
FROM Intervenant
WHERE salaireJ = (
    SELECT MAX(salaireJ)
    FROM Intervenant
);
```

-
10. Déterminer le coût des interventions faites sur la parcelle de désignation "Parcelle2" depuis le début de l'offre des services de l'entreprise agricole pour cette parcelle.

```

SELECT SUM(nbJours*salaireJ)
FROM Parcelle P, Intervenant I, Intervention It
WHERE P.idPar =It.idPar
AND I.idIn=It.idIn
AND designation LIKE "Parcelle2";

```

11. Déterminer les désignations des parcelles qui n'ont subi aucune intervention.

```

SELECT designation
FROM Parcelle
WHERE idPar IN ( SELECT idPar
                  FROM Parcelle
                  EXCEPT
                  SELECT idPar
                  FROM Intervention
                );

```

12. Déterminer le nombre des interventions réalisées par mois durant l'année 2020.

Indication

On donne la fonction `strftime` pouvant être invoquée dans une requête SQL et ayant la syntaxe suivante :

`Strftime(format,date)` qui pour une date identifiée par `date` et un format `format` (comme illustré dans le tableau ci-dessous) retourne une chaîne de caractères comme le montre l'exemple.

Format	Signification
%d	Jour du mois sur 2 caractères de "01" à "31"
%m	Mois de l'année sur 2 caractères de "01" à "12"
%y	L'année sur 4 caractères.

Exemple :

```

strftime("%d","2021-06-26") retourne "26"
strftime("%m","2021-06-26") retourne "06"
strftime("%Y","2021-06-26") retourne "2021"

```

```

SELECT strftime ("%m", dateDebut) AS mois, COUNT(*)
FROM Intervention
WHERE strftime ("%Y", dateDebut)="2020"
GROUP BY mois;

```

Partie 3

Dans la suite les fonctions demandées doivent être écrites en Python. On désigne par **Cur** le curseur d'exécution de requêtes et **dico** un dictionnaire tel que chaque clé est un numéro de mois et chaque valeur est le nombre de jours pour ce mois. On ne gère pas les années bissextiles.

```
dico={1:31,2:28,3:31,4:30,5:31,6:30,7:31,8:31,9:30,10:31,11:30,12:31}
```

-
1. Écrire une fonction **salaire_jour** qui prend en paramètre **Cur** et l'identifiant **Id** d'un intervenant, retourne son salaire journalier.

```
def salaire_jour(Cur, Id):
    Cur.execute("""
    SELECT salaireJ
    FROM Intervenant
    WHERE idIn LIKE '{}'
    """.format(Id))
    return Cur.fetchone()[0]
```

-
2. Écrire une fonction **t_precedent** qui prend en paramètre un mois **M** et une année **A**, retourne un tuple contenant le mois précédant **M** ainsi que l'année correspondante.

```
def t_precedent(M, A):
    if 2 <= M <= 12:
        M=M-1
    else:
        M=12
        A=A-1
    return (M,A)
```

-
3. Écrire une fonction **salaire_mois** qui prend en paramètre **Cur**, un mois **M**, une année **A** et l'identifiant **Id** d'un intervenant, retourne son salaire mensuel s'il a effectué des interventions et 0 sinon.

```
def salaire_mois(Cur,M, A , Id):
    t=t_precedent(M,A)
    k1=''
    if len(str(M))==1:
        k1='0'
    k2=''
    if len(str(t[0]))==1:
        k2='0'
    Cur.execute("""
    SELECT strftime('%d',dateDebut),
           strftime('%m',dateDebut),
           strftime('%Y',dateDebut),
           nbJours
```

```

        FROM Intervenant I, Intervention It
        WHERE I.idIn=It.idIn
        AND dateDebut>='{}-{}-01'
        AND dateDebut<='{}-{}-{}'
        AND I.idIn={}""".format(t[1],k2,t[0],A,k1,M,dico[M],idIn))
L=Cur.fetchall()
nj=0
for i in L:
    if int(i[1])==M:
        if int(i[0])+i[3]<=dico[M]:
            nj+=i[3]
        else:
            nj+=dico[M]-int(i[0])+1
    else:
        if int(i[0])+i[3]>dico[t[0]]:
            nj+=int(i[0])+i[3]-dico[t[0]]-1
tf=salaire_jour(Cur,Id)
return nj*tf

```

-
4. Écrire une fonction **salaire_an** qui prend en paramètre **Cur**, une année **A** et l'identifiant **Id** d'un intervenant, retourne son salaire annuel.

```

def salaire_an(Cur, A , Id):
    L=[]
    for M in range(1,13):
        L.append(salaire_mois(Cur, M, A , Id))
    return(sum(L))
#version2
def salaire_an(Cur, A , Id):
    return sum(salaire_mois(Cur, M, A , Id) for M in range(1,13))

```

-
5. Ecrire le script python permettant de :

- importer le module **sqlite3** ;
- se connecter à la base de données '**BDAgricole.db**' ;
- créer le curseur **cur** d'exécution ;
- afficher les salaires annuels de tous les intervenants de l'entreprise agricole pour l'année 2020.

```

import sqlite3
BD=sqlite3.connect('BDAgricole.db')
Cur=BD.cursor()
dico={1:31,2:28,3:31,4:30,5:31,6:30,7:31,8:31,9:30,10:31,11:30,12:31}
Cur.execute("select idIn from Intervenant")
L=Cur.fetchall()
for id in L:
    print("id={} salaire annuel={} ".format(id[0], salaire_an(Cur,
2020 , id[0])))

```

ANNEXE – Quelques Fonctions/Méthodes Python

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

Module numpy

- **M.shape** ou **shape(M)** retourne un tuple formé par le nombre de lignes et le nombre de colonnes d'une matrice M.
- **zeros(...)** retourne un tableau initialisé par des 0.
- **ones(...)** retourne un tableau initialisé par des 1.
- **M.copy()** retourne une copie de M.
- **M.min()** ou **min(M)** et **M.max()** ou **max(M)** retournent respectivement la valeur minimale et la valeur maximale d'une matrice M.
- **np.all(M)** retourne `True` si tous les éléments de M valent `True` et `False` sinon.
- **np.any(M)** retourne `True` si M contient au moins un élément qui vaut `True` et `False` sinon.

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.
- **min(it)** retourne la valeur minimale de l'itérable it.
- **max(it)** retourne la valeur maximale de l'itérable it.
- **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.
- **lst.sort()** trie la liste lst dans l'ordre croissant.
- **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.index(val)** retourne l'indice de la première occurrence val de la liste lst.
- **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.
- **s.add(obj)** ajoute obj à un ensemble s (instance de la classe `set`).
- **s.remove(obj)** supprime obj de l'ensemble s (instance de la classe `set`).
- **s1.union(s2)** retourne l'ensemble union des deux ensembles s1 et s2.
- **s1.intersection(s2)** retourne l'ensemble intersection des deux ensembles s1 et s2.
- **s1.difference(s2)** retourne l'ensemble différence des deux ensembles s1 et s2.