

Quelques examens et DS d'Informatique avec corrigés

Pour la préparation de l'épreuve d'Informatique du
Concours National d'Entrée aux Ecoles d'Ingénieurs
MP-PC-T 2020



Collectés par :

Mohamed Hammami (IPEIEM)

Wafa Meftah (IPEIS)

Houcemeddine Filali (IPEIM)

Laila Hammami (IPEIN)

Hayet Latrach (IPEIT)

Besma Fayech (ENSIT)



Devoir surveillé Numéro 2

Matière Informatique	Nombre de pages 5	nombre d'heures 1 h	Classes 2 ^{ème} Cycle préparatoire
Enseignant		Houcemeddine FILALI	

NB : Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Problème - Décomposition de Choleskey (20 points)

L'objectif de ce problème est de résoudre un système d'équations linéaire déterminé de la forme :

$$Ax = b \quad (1)$$

où $A \in M_n(\mathbb{C})$ est une matrice Hermitienne définie positive et $b \in \mathbb{C}^n$ un vecteur à composantes complexes pour $n \in \mathbb{N}^*$.

- (1) Donner les instructions Python permettant d'importer le module `numpy` ainsi que les fonctions de l'algèbre linéaire.

Solution:

```
import numpy as np
from numpy import linalg
```

Une matrice $A \in M_n(\mathbb{C})$ est dite Hermitienne (ou auto adjointe) si et seulement si elle vérifie la relation décrite par l'équation suivante :

$$A = \overline{A}^T \Leftrightarrow a_{ij} = \overline{a_{ji}} \quad \forall i, j \in \{1 \dots n\} \quad (2)$$

où $\overline{A}^T$ est la matrice transconjugée de A (matrice transposée de la matrice conjuguée).

Par exemple, la matrice $A = \overline{A}^T = \begin{pmatrix} 1 & 1+2i & 3i \\ 1-2i & 2 & 1+i \\ -3i & 1-i & 5 \end{pmatrix}$ est Hermitienne.

- (2) Écrire une fonction `is_hermitian` qui prend en entrée une matrice a (instance de la classe `ndarray`) et qui retourne un `bool` (`True` si a est Hermitienne et `False` sinon).

Solution:

```
def is_hermitian(a):
    return np.all(a == a.T.conjugate())
```

Une matrice est dite définie positive si et seulement si toutes ses valeurs propres sont strictement positives.

- (3) Écrire une fonction `is_positive_definite` qui prend en entrée une matrice a (instance de la classe `ndarray`) et qui retourne un `bool` (`True` si a est définie positive et `False` sinon).

Solution:

```
def is_positive_definite(a):
    return np.all(linalg.eigvals(a) > 0)
```

La décomposition de **Choleskey** permet de décomposer une matrice $A \in M_n(\mathbb{C})$ hermitienne définie positive en un produit faisant intervenir une autre matrice $L \in M_n(\mathbb{C})$ triangulaire inférieure « Lower triangular » tel que :

$$A = L \bar{L}^T \quad (3)$$

Où $\bar{L}^T$ est la matrice transconjugée de L . Les coefficients l_{ij} de la matrice L sont décrits par les équations suivantes :

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} \bar{l}_{jk}}{l_{jj}} \quad \forall i, j \text{ tel que } n \geq i > j \geq 1 \quad (4)$$

$$l_{ij} = 0 \quad \forall i, j \text{ tel que } n \geq j > i \geq 1 \quad (5)$$

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik} \bar{l}_{ik}} \quad \forall i \in \{1 \dots n\}. \quad (6)$$

Le calcul des éléments de l doit s'effectuer dans l'ordre ligne par ligne ou bien colonne par colonne en utilisant les équations 4, 6 et 5.

3 points

- (4) Écrire une fonction **choleskey** qui prend en entrée une matrice a (instance de la classe **ndarray**) supposée hermitienne et définie positive et qui retourne en sortie la matrice l (instance de la classe **ndarray**) qui vérifie la relation décrite par l'équation 3.

Solution:

```
def choleskey(a):
    l = np.zeros(a.shape, np.float)
    h, w = a.shape
    for i in range(h):
        for j in range(i):
            l[i,j] = (a[i,j] - l[i,:j].dot(l[j,:j].conjugate()))/l[j,j]
        l[i,i] = np.sqrt(a[i,i] - l[i,:i].dot(l[i,:i].conjugate()))
    return l
```

Un système d'équation linaires est dit triangulaire supérieur s'il a la forme suivante :

$$\begin{cases} u_{11}x_1 + u_{12}x_2 + \dots + u_{1n}x_n = b_1 \\ u_{22}x_2 + \dots + u_{2n}x_n = b_2 \\ \vdots \\ u_{nn}x_n = b_n \end{cases} \quad (7)$$

Ce système peut s'écrire sous la forme matricielle suivante :

$$U x = b \text{ où } U = \begin{pmatrix} u_{11} & \dots & u_{1n} \\ 0 & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 & u_{nn} \end{pmatrix} \in M_n(\mathbb{C}) \quad (8)$$

est une matrice triangulaire supérieure et $b \in \mathbb{C}^n$. La résolution d'un système d'équations de forme triangulaire supérieure, peut s'effectuer itérativement du bas vers le haut, en remontant et en remplaçant les x_i pour i allant de n vers 1 par les solutions trouvées dans l'étape précédente.

$$x_i = \frac{b_i - \sum_{k=i+1}^n u_{ik}x_k}{u_{ii}} \quad \forall i \text{ allant de } n \rightarrow 1 \quad (9)$$

3 points

(5) Écrire une fonction `solve_upper` qui prend en entrée :

- Une matrice `u` de forme triangulaire supérieure.
- un vecteur `b`.

La fonction doit résoudre le système puis renvoyer le vecteur solution x en appliquant le procédé de remonté expliquée ci-haut.

Solution:

```
def solve_upper(u,b):
    x = np.ndarray(b.shape)
    for i in reversed(range(b.size)):
        x[i] = (b[i]-u[i,i+1:].dot(x[i+1:]))/u[i,i]
    return x
```

Un système triangulaire inférieur est décrit par la forme matricielle suivante :

$$L x = b \text{ où } L = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n1} & \dots & \dots & l_{nn} \end{pmatrix} \in M_n(\mathbb{C}) \quad (10)$$

une matrice triangulaire inférieure et $b \in \mathbb{C}^n$.

3 points

(6) Écrire une fonction python `solve_lower` qui prend en entrée :

- Une matrice `l` de forme triangulaire inférieure.
- Un vecteur `b`.

La fonction doit renvoyer x la solution du système $l x = b$ en appliquant l'équation suivante :

$$x_i = \frac{b_i - \sum_{k=1}^{i-1} l_{ik}x_k}{l_{ii}} \quad \forall i \text{ allant de } 1 \rightarrow n.$$

Solution:

```
def solve_lower(l,b):
    x = np.ndarray(b.shape)
    for i in range(b.size):
        x[i] = (b[i]-l[i,:i].dot(x[:i]))/l[i,i]
    return x
```

Étant donnée une matrice $A \in M_n(\mathbb{C})$ supposée hermitienne et définie positive soit $L \in M_n(\mathbb{C})$ la matrice triangulaire inférieure résultante de sa décomposition de Choleskey $A = L\bar{L}^T$. Il est possible de transformer la résolution du système $Ax = b$ en une résolution de deux systèmes linéaires (*un système inférieur puis un système supérieur*)

$$Ax = b \Leftrightarrow \begin{cases} L y = b \\ \bar{L}^T x = y \end{cases} \quad \text{et}$$

3 points

(7) Écrire une fonction `solve_choleskey` qui prend en entrée :

- a une matrice (instance de la classe `ndarray`) supposée hermitienne définie positive.
- b un vecteur (instance de la classe `ndarray`)

La fonction doit renvoyer x la solution du système linéaire $ax = b$ en appliquant démarche décrite ci-haut.

Solution:

```
def solve_choleskey(a,b):
    l = choleskey(a)
    lt = l.T.conjugate()
    y = solve_lower(l,b)
    return solve_upper(lt,y)
```

2 points

(8) Analyser la complexité asymptotique de la fonction `solve_choleskey`.

Solution:

$$O(a.size^{\frac{3}{2}})$$

Annexe

Le tableau suivant contient quelques commandes utiles que l'étudiant peut éventuellement utiliser pour la composition de ses réponses.

Commande Python	Description
<code>import numpy as np</code>	Importation du module numpy.
<code>z = np.array([[1, 1+2j, 3j], [1-2j, 2, 1+1j], [-3j, 1-1j,5]])</code>	Création d'une matrice z contenant des coefficients complexes.
<code>np.identity(n)</code>	Renvoie la matrice identité d'ordre n sous forme d'un ndarray à deux dimensions.
<code>z.T</code> ou bien <code>z.transpose()</code>	Donne la matrice transposée de z.
<code>z.conjugate()</code>	Produit une matrice formée par les conjugués des éléments de z (terme à terme).
<code>np.all(a)</code>	Donne True si tous les éléments du ndarray a valent True , sinon renvoie False .
<code>np.any(a)</code>	Renvoie True si le ndarray a contient au moins une valeur True , sinon False .
<code>a[dep :fin :pas]</code>	Permet d'extraire une partie du vecteur a (ndarray) formé par les éléments d'indices compris entre dep (inclue) et fin (exclue) et séparés par la valeur pas.
<code>a.dot(b)</code> ou bien <code>a@b</code>	Renvoie le produit scalaire de a et b (pour a et b naddays à une seule dimension). Sinon elle renvoie le résultat du produit matriciel entre a et b.
<code>a.sum()</code>	Renvoie la somme de tous les éléments du ndarray a.
<code>a.sum(axis = i)</code>	Renvoie un ndarray qui résulte de la somme des sous vecteurs projetés sur l'axe numéro i.
<code>a.max()</code>	Renvoie la valeur maximale du ndarray a.
<code>a.argmax()</code>	Renvoie l'indice de la valeur maximale du ndarray a.
<code>np.where(a > c)</code>	Renvoie les indices où les éléments du ndarray a sont plus grand que c.
<code>from numpy import linalg</code>	Importation du module linalg contenant les fonctionnalités de l'algèbre linéaire.
<code>linalg.eig(a)</code>	Renvoie un tuple formé par deux vecteurs vp et p, tel que : vp contient les valeurs propres de a et p contient les vecteurs propres de a.
<code>linalg.eigvals(a)</code>	Renvoie un vecteur à une seule dimension (ndarray) contenant les valeurs propres de a.
<code>linalg.det(a)</code>	Renvoie le déterminant de a.

 INSTITUT PREPARATOIRE AUX ETUDES D'INGÉNIEURS DE MONASTIR المعهد التحضيري للدراسات الهندسية بالمنستير		Année universitaire 2017	
Examen N°2			
Matière Informatique	Nombre de pages 10	nombre d'heures 2 h	Classes 2 ^{ième} Cycle préparatoire
Enseignant		Houcemeddine FILALI	

NB : Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Problème

20(points)

Dans ce problème on s'intéresse à l'implémentation d'un algorithme de détection de contours appliqué sur les images matricielles en niveau de gris. On rappelle qu'une image en niveau de gris I est représentée en **PYTHON** par une instance de la classe `ndarray` et que la propriété `I.shape` de cette instance renvoie un `tuple` (h, w) contenant respectivement la hauteur et la largeur de l'image. Les valeurs `I[x,y]` sont des entiers (`np.uint8`) ou des décimaux (`np.float`) proportionnels à l'intensité lumineuse au niveau du pixel de coordonnées (x,y).

La détection des contours est un des problèmes les plus fondamentaux et les plus importants dans le domaine de traitement des images numériques. Les contours sont des points de l'image (**PIXELS**) caractérisés par un fort contraste (un saut substantiel en intensité lumineuse). Un algorithme de détection de contours produit en sortie une image binaire IC tel que ($IC_{xy} \in \{0, 1\} \forall x, y$) appelée carte des contours où les points blancs marquent les frontières entre les régions de l'image. Ainsi, la détection des contours permet de réduire les données de l'image tout en préservant les propriétés structurales de cette dernière. On suppose que les réponses des questions suivantes seront placées dans un script python commençant par les lignes suivantes :

```
import numpy as np
import matplotlib.pyplot as plt
```

- 1) (1 point) Écrire une fonction `load_image(path)`. Cette fonction doit lire le fichier image dont le chemin d'accès est spécifié par la `str path` passée en entrée et doit retourner une matrice I (instance de la classe `ndarray`) formée par des nombres décimaux (`np.float`) tels que ($0 \leq I_{ij} \leq 1$). Si l'image lue à partir du fichier est formée par des `np.uint8` alors elle est divisée par 255. Si l'image lue à partir du fichier est une image couleur elle est convertie en une image niveau de gris. On rappelle qu'une image couleur est représentée par un `ndarray I` tel que `I.shape` renvoie (h,w,p) où :

- h est la hauteur de l'image.
- w est la largeur de l'image
- p est la profondeur de l'image (3 pour les images couleurs codées en rvb)

La relation suivante permet de calculer le niveau de gris g dans un pixel couleur codé en rvb :

$$g = 30\%r + 59\%v + 11\%b$$

Solution:

```
def load_image(path):
    img = plt.imread(path)
```

```

if img.dtype == np.uint8:
    img = img / 255
if img.ndim == 3: #couleurs
    img = (img * np.array([0.3, 0.59, 0.11])).sum(-1)
return img

```

L'opération de filtrage linéaire correspond à une convolution de la matrice de l'image **IS** par un masque **M** (une matrice de taille $n \times n$), cette opération produit une autre image **IR** de même taille que **IS**. Dans ce problème, on considère uniquement les masques carrés tel que n est impair. La table 1 illustre un exemple de masque de taille 3×3 . Noter que les indices d'un masque représentent le décalage par rapport au point central du masque M_{00} . Ainsi, pour le masque illustré dans le tableau 1, la première ligne possède l'indice -1, la deuxième l'indice 0 (la ligne du point central), la troisième ligne possède l'indice 1. Idem, pour les colonnes. La convolution consiste à coulisser la matrice **M** sur les pixels de l'image source **IS**_{*xy*} pour toutes les valeurs possibles de *x* et de *y*. L'application du masque **M** sur **IS**_{*xy*} produit une autre image **IR** tel que :

$$IR_{xy} = \sum_{l=-n \div 2}^{n \div 2} \sum_{c=-n \div 2}^{n \div 2} M_{lc} * IS_{x+l \ y+c}, \text{ où } x \div y \text{ dénote le quotient de la division euclidienne de } x \text{ par } y. \quad (1)$$

Cette opération est illustrée dans la figure 1.

$m_{-1 \ -1}$	$m_{-1 \ 0}$	$m_{-1 \ 1}$
$m_{0 \ -1}$	$m_{0 \ 0}$	$m_{0 \ 1}$
$m_{1 \ -1}$	$m_{1 \ 0}$	$m_{1 \ 1}$

TABLE 1 – Masque m de taille 3×3

Dans l'équation précédente un problème se manifeste aux bords de l'image **IS**, particulière au niveau des points du masque où **IS**_{*x+l y+c*} n'est pas définie. Par exemple l'application d'un masque sur le point **IS**₀₀ manifeste une singularité (car la ligne -1 et la colonne -1 ne sont pas définies). Pour résoudre ce problème on convient de considérer l'image **IS** comme un tore (voir figure 2) qu'on pourra balayer indéfiniment, ainsi, les problèmes aux bords ne se posent plus. Quand on dépasse la largeur de l'image, le parcours reprend à partir de la première ligne. Idem, pour les colonnes. La formule décrite par l'équation 1 est remplacée par la formule suivante :

$$IR_{xy} = \sum_{l=-n \div 2}^{n \div 2} \sum_{c=-n \div 2}^{n \div 2} M_{lc} * IS_{x'y'}, \text{ où } \begin{cases} x' = (x + l) \mod h & : h \text{ la hauteur de l'image IC} \\ y' = (y + c) \mod w & : w \text{ la largeur de l'image IC.} \end{cases} \quad (2)$$

Dans la suite on propose de concevoir une classe **Mask** pour représenter la matrice d'un filtre linéaire. Développer le code de cette classe en respectant les consignes suivantes :

- 2) (2 points) Un constructeur qui prend en entrée **a** un **ndarray** à deux dimensions de même taille n , où n est un nombre impair. *a* doit être formée par des nombres décimaux (**np.float**). Le constructeur doit faire les vérifications nécessaires, déclencher des exceptions le cas échéant et créer un attribut d'instance privé contenant une copie de ce **ndarray**. L'étudiant peut délibérément attacher d'autres attributs d'instance (ajouter des commentaires explicatifs dans ce cas).
- 3) (1 point) Définir la méthode magique **__getitem__** qui prend en entrée un **tuple** **t** formé par une paire d'entiers (*l,c*) et qui retourne la valeur de l'élément de self décalé de *l* lignes et de *c* colonnes par rapport au point central du masque (voir table 1). Exemple :

```

>>>M = Mask(np.arange(9).reshape((3,3)))
>>>M[0,0] # donne 4
>>>M[-1,-1]# donne 0

```

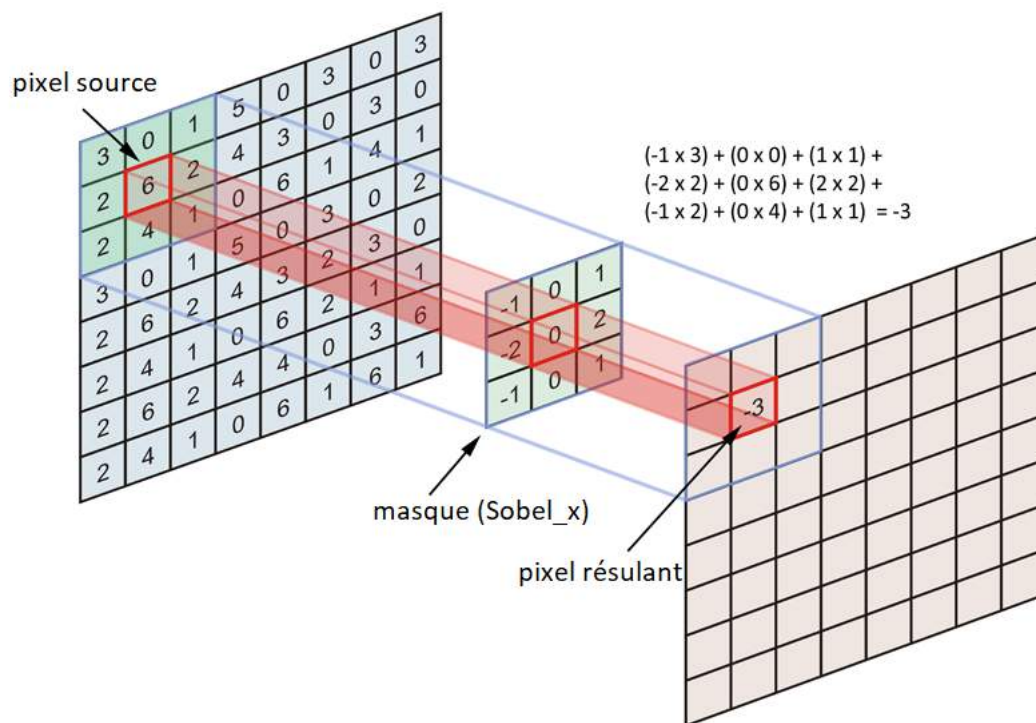


FIGURE 1 – Illustration d'un produit de convolution

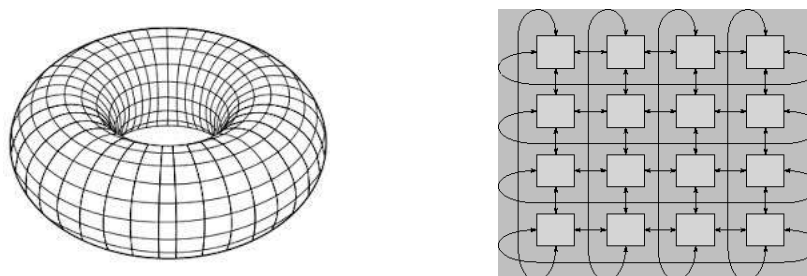


FIGURE 2 – Forme torique (à gauche), parcours d'une matrice torique (à droite)

- 4) (1 point) Définir la méthode magique (méthode d'instance) `__setitem__` qui prend en entrée :
- un tuple `t` formé par une paire d'entiers `(l,c)`.
 - un float `nv`.

La méthode permet de remplacer l'ancienne valeur de `M[l,c]` par `nv`.

- 5) ($\frac{1}{2}$ point) Définir l'attribut `edge` : un entier indiquant la taille d'un côté du masque `self`. Par exemple, pour le masque `M` précédent : `M.edge` doit retourner 3.
- 6) (1 point) Définir la méthode `indices` qui retourne un ensemble (une instance de la classe `set`) de tuples `(i,j)` contenant les indices de tous les éléments du masque `self`. Par exemple, pour le masque défini dans la question précédente :

```
>>>M.indices # donne
{(0, 1), (-1, 1), (0, 0), (-1, 0), (-1, -1), (0, -1), (1, 0), (1, -1), (1, 1)}
```

Ainsi, on pourra parcourir les éléments d'un masque de la manière suivante :

- 7) ($\frac{1}{2}$ point) Soit `m` une instance de la classe `Mask`, établir une relation entre `len(e.indices)` et `e.edge`.

Solution:

$$\text{len}(e.\text{indices}) = e.\text{edge}^2$$

Produit cartésien d'un ensemble avec lui-même.

- 8) (2 points) Définir la méthode magique `__mul__(self, other)` où `other` est une matrice (instance d'un `ndarray`) supposée retournée par la fonction `load_image`. Cette méthode doit retourner une autre matrice contenant le résultat de convolution du masque `self` sur l'image `other`, utiliser l'équation 2.

Ainsi, pour appliquer un masque sur une image il suffit d'appliquer l'opérateur de multiplication `*` en mettant l'image à droite de ce dernier. `M * I` permet de convoluer l'image `I` par le masque `M`.

- 9) (1 point) Exprimer la complexité asymptotique de votre implémentation en fonction de `self.edge` et `other.shape`.

Solution:

$$O(\text{self.edge}^2 \times \text{other.shape}[0] \times \text{other.shape}[1])$$

Solution:

```
class Mask:
    def __init__(self, a):
        assert isinstance(a,np.ndarray)
        assert a.ndim == 2
        assert a.shape[0] == a.shape[1]
        assert a.shape[0] % 2 == 1
        assert a.dtype == np.float
        self.weights = a.copy()
        self.edge = a.shape[0]
        self.disp = a.shape[0]//2

    def __getitem__(self, t):
        l, c = t
        return self.weights[l+self.disp,c + self.disp]
```

```

def __setitem__(self, t, nv):
    l, c = t
    self.weights[l+self.disp, c + self.disp] = nv

def indices(self):
    return {(i,j) for i in range(-self.disp, self.disp+1)
            for j in range(-self.disp, self.disp+1)}

def __mul__(self, other):
    ir = np.zeros(other.shape)
    h, w = ir.shape
    for x in range(h):
        for y in range(w):
            for l, c in self.indices():
                xp = (x+l) % h
                yp = (y+c) % w
                ir[x,y] += self[l,c] * other[xp, yp]

    return ir

```

La méthode de **Canny**, aussi connue comme détecteur de contours optimal applique la démarche décrite ci-dessous afin d'extraire la carte des contours IC à partir d'une image source IS en niveaux de gris :

- Le bruit est atténué en appliquant un filtre gaussien sur l'image source IS. Ce filtre produit un effet de flou sur l'image permettant ainsi de réduire le nombre de fausses détections.
 - Deux images G_x et G_y contenant les gradients horizontaux $\frac{\partial IS}{\partial x}$ et verticaux $\frac{\partial IS}{\partial y}$ sont calculés par application des filtres de Sobel. Les gradients permettent de détecter les points de haute variation spatiale qui forment les points potentiels du contour recherché.
 - Deux autres images G_ρ et G_θ sont calculées à partir de G_x et G_y . Ces deux premières contiennent respectivement, la norme et l'angle du gradient en chaque pixel (x,y) de l'image. L'image G_ρ contient une première approximation de la carte des contours.
 - La carte des directions du gradient G_{dir} est calculée à partir de G_θ . La carte G_ρ est raffinée en mettant à zéros les points $G_{\rho_{xy}}$ qui ne sont pas maximaux par rapport à la direction du gradient.
 - Finalement, l'image G_ρ sont binarisé par un double seuillage avec hystérésis.
- 10) (1 point) Écrire une fonction `gaussian(n, sigma)` qui prend deux entrées :
- `n` un entier positif impair.
 - `sigma` un `float`.

La fonction retourne une instance de la classe **Mask** contenant un filtre gaussien de taille `n` et de variance `sigma`. Les éléments de ce masque sont calculés par la fonction suivante où (x,y) désignent le décalage par rapport à l'élément central du masque :

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp -\frac{x^2 + y^2}{2\sigma^2}$$

Solution:

```

def gaussian(n,sigma):
    res = Mask(np.zeros((n,n)))

```

```

for x, y in res.indices():
    res[x,y] = 1/(2*np.pi*sigma**2) * np.exp(-(x**2+y**2)/(2*sigma**2))
return res

```

- 11) (1 point) Les filtres de Sobel sont des filtres dérivateurs de taille 3×3 permettant de calculer les gradients horizontaux et verticaux d'une image. Les éléments du masque de Sobel pour les gradients horizontaux sont illustrés par la matrice

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}.$$

$S_y = -t_{S_x}$ (où t_{S_x} est la matrice transposée de S_x) permet d'obtenir les gradients verticaux .

Écrire une fonction `sobel` qui prend en entrée un paramètre optionnel `vertical` (avec `False` comme valeur par défaut). La fonction doit retourner une instance de la classe `Mask` contenant le masque de S_x si `vertical` vaut `False` et S_y sinon.

Solution:

```

def sobel(vertical = False):
    sx = np.array([[-1,0,1],[-2, 0, 2], [-1,0,1.0]])
    return Mask(sx) if not vertical else Mask(-sx.T)

```

- 12) (1 point) Écrire une fonction `polar_grad` qui prend en entrée un `ndarray` à deux dimensions `I` représentant une image en niveaux de gris. La fonction doit retourner un `tuple` contenant deux instances de la classe `ndarray` `G_rho` et `G_theta` tel que :

$$\left\{ \begin{array}{ll} G_\rho = \sqrt{G_x^2 + G_y^2} & \text{l'image de la norme du gradient.} \\ G_\rho = \frac{G_\rho}{\text{valeur maximale de } G_\rho} & \text{on normalise la norme du gradient pour avoir des valeurs dans } [0, 1] \\ \text{et} & \\ G_\theta = \text{np.arctan2}(G_y, G_x) & \text{l'image de l'angle du gradient définie sur } [-\pi, \pi]. \end{array} \right.$$

Où G_x et G_y sont les images des gradients horizontaux et verticaux obtenues respectivement par la convolution de l'image I par les masques S_x et S_y .

Solution:

```

def polar_grad(img):
    sx = sobel()
    sy = sobel(True)
    Gx = sx * img
    Gy = sy * img
    G_rho = np.sqrt(Gx**2 + Gy**2)
    G_theta = np.arctan2(Gy, Gx)
    G_rho = G_rho / G_rho.max()
    return G_rho, G_theta

```

- 13) (1 point) Dans ce problème on s'intéresse aux 4 directions numérotées de 0 à 3 comme illustré par le tableau ci-dessous. Le carré contenant la boussole représente le pixel central.

3	2	1
		
0	2	0
1	2	3

Pour obtenir la direction à partir d'un angle $\theta \in [-\pi, \pi]$, on applique la transformation suivante sur l'image G_θ :

1. $G_{\text{rot}} = ((G_\theta + \frac{\pi}{8}) \bmod \pi)$.
2. Ensuite, on crée un vecteur **zones** par discrétisation de l'intervalle $[\frac{\pi}{4}, \pi]$ en 4 morceaux équidistants.
3. L'image de la direction du gradient G_{dir} est calculée comme suit :

$G_{\text{dir}}[x,y] = i$, où i est le plus petit entier vérifiant $G_{\text{rot}}[x,y] \leq \text{zones}[i]$.

Écrire une fonction **direction** qui prend en entrée l'image de l'angle du gradient G_θ et qui retourne, l'image G_{dir} contenant des entiers dans 0,3 décrivant la direction du gradient.

Solution:

```
def direction(G_theta):
    G_rot = (G_theta + np.pi / 8) % np.pi
    zones = np.linspace(np.pi/4, np.pi, 4)
    def f(val):
        for i in range(zones.shape[0]):
            if val <= zones[i]:
                return i
    f = np.vectorize(f)
    return f(G_rot)
```

- 14) (1 point) Les deux tableaux suivants indiquent les déplacements dx et dy qu'il faut appliquer à un pixel de coordonnées $p = (x,y)$ pour aller dans chacune des 4 directions.

	-1	0	1			-1	-1	-1
								
	-1		1			0		0
	-1	0	1			1	1	1

Écrire deux fonctions $dx(\text{dir})$ et $dy(\text{dir})$ qui prennent en entrée un entier $\in \{0, 1, 2, 3\}$ indiquant la direction du gradient et qui retournent un **tuple** d'entiers contenant les déplacements qu'il faut appliquer sur x respectivement y pour aller dans la direction indiquée par le paramètre **dir**.

Par exemple $dx(0)$ donne $(-1, 1)$ $dx(1)$ donne $(-1, 1)$ et $dx(2)$ donne $0, 0$ etc.

Solution:

```
def dx(dir):
    r = {0:(-1,1), 1:(-1,1), 2:(0,0),3:(1,-1)}
    return r[dir]

def dy(dir):
    r = {0:(0,0), 1:(1,-1), 2:(1, -1), 3:(1,-1)}
    return r[dir]
```

- 15) (1 point) Écrire une fonction **suppression** qui prend en entrée une image **G_rho** contenant la norme du gradient, une autre image **G_dir** contenant la direction du gradient. La fonction doit retourner une image **IC_1** tel que :

- $IC_1[x,y] = G_rho[x,y]$ si $G_rho[x,y] \geq \max(G_rho[xp1, yp1], G_rho[xp2, yp2])$ pour $(xp1, yp1)$ et $(xp2, yp2)$ deux positions voisines dans la direction donnée par $G_dir[x,y]$.
- $IC_1[x,y] = 0$ sinon.

Exemple : pour $G_dir[x,y] = 0$ (direction horizontale) et $G_rho =$

0.8	0.7	0.3
0.8	0.8 (le pixel x,y)	0.7
0	0.9	0.1

$IC[x,y]$ contiendra 0.8 car G_rho est maximal dans sa direction. Par contre si $G_dir[x,y] = 2$ (direction verticale) alors $IC[x,y]$ contiendra 0 car G_rho n'est pas maximal dans sa direction.

Solution:

```
def suppression(g_rho, g_dir):
    ic_1 = g_rho.copy()
    h, w = ic_1.shape
    for x in range(h):
        for y in range(w):
            dir = g_dir[x,y]
            dx1, dx2 = dx(dir)
            dy1, dy2 = dy(dir)
            xp1, xp2 = (x+dx1) % h, (x+dx2) % h
            yp1, yp2 = (y+dy1) % w, (y+dy2) % w
            if ic_1[x,y] < max(g_rho[xp1, yp1], g_rho[xp2, yp2]):
                ic_1[x,y] = 0
    return ic_1
```

- 16) (1 point) Écrire une fonction **double_threshold** qui prend en entrée :

- **IC_1** (une image contenant le module du gradient raffiné par l'application de la fonction précédente)
- **T_sup** un seuil supérieur au-dessus duquel on peut confirmer avec sûreté que le pixel $IC_1[x,y]$ est un point contour.
- **T_inf** un seuil inférieur au-dessous duquel on peut confirmer avec sûreté que le pixel $IC_1[x,y]$ n'est pas un point contour.

La fonction retourne une image **IC_2** ayant les mêmes dimensions que **IC_1** tel que :

- $IC_2[x,y] = 1$ si $IC_1[x,y] \geq T_sup$.
- $IC_2[x,y] = 0.5$ si $T_inf \leq IC_1[x,y] < T_sup$.
- $IC_2[x,y] = 0$ si $T_inf > IC_1[x,y]$.

Solution:

```
def double_thershold(ic_1, t_sup, t_inf):
    msk1 = ic_1 >= t_sup
    msk2 = ic_1 < t_inf
    ic_2 = np.zeros(ic_1.shape)
    ic_2[msk1] = 1
    ic_2[msk2] = 0
    ic_2[~msk1&~msk2] = 0.5
    return ic_2
```

17) (1 point) Écrire une fonction récursive **closure** qui prend en entrée :

- I une image de taille $h \times w$.
- v = une matrice de booléens ayant la même taille que I , où $v[x,y]$ indique si le pixel (x,y) a été traité ou non. ($v[x,y] == \text{True}$ implique que $I[x,y]$ a été traité).
- $t = (x,y)$ un **tuple** indiquant les coordonnées d'un pixel dans l'image.

La fonction doit retourner un ensemble E contenant les coordonnées des pixels p tel que :

- $\forall p \in E \quad I[p] \neq 0$.
- $\forall p \in E \quad \exists (p_0, p_1, \dots, p_k) \in E$. où p_i est voisin de $p_{i+1} \forall i < k$ et p_k est voisin à p et p_0 voisin à t .

pour ce faire, la fonction doit se comporter comme suit :

- si $I[x,y] = 0$ ou si (x,y) est déjà traité alors retourner l'ensemble vide
- sinon :
 - marquer $v[x,y]$ comme pixel traité
 - créer un singleton de type **set** $e = \{(x,y)\}$
 - pour chaque pixel (xv, yv) dans les 8 voisins immédiats de (x,y) : si $I[xv, yv]$ est non nul alors faire un appel récursif **connect**($I, v, (xv, yv)$) récupérer le résultat de l'appel dans un ensemble r , et mettre à jour e de sorte que $e \leftarrow e \cup r$.

Solution:

```
def closure(I, v, t):
    x, y = t
    h, w = I.shape
    if I[x,y] == 0 or v[x,y]:
        return set()
    else:
        v[x,y] = True
        e = {(x,y)}
        m = Mask(np.zeros((3,3)))
        for l, c in m.indices():
            xv, yv = (x+l) % h, (y+c)%w
            if I[xv, yv] != 0:
                r = closure(I, v, (xv, yv))
                e |= r
        return e
```

18) (1 point) En utilisant la fonction précédente, écrire une fonction **edge_map** qui prend en entrée une image IC_2 contenant des valeurs de l'ensemble $\{0, 0.5, 1\}$ supposée résultante de la fonction précédente **double_threshold**. La fonction **edge_map** doit produire en sortie une image binaire IC (contenant des zéros et des uns uniquement) telle que :

- $IC[x,y] = 0$ si $IC_2[x,y] == 0$.
- $IC[x,y] = 1$ si $IC_2[x,y] == 1$.
- Si $IC_2[x,y] == 0.5$ alors :
 - $IC[x,y] = 1$ s'il existe un chemin formé par de valeurs toutes non nulles connectant (x,y) à un pixel (xp, yp) tel que $IC_2[xp,yp] == 1$.
 - $IC[x,y] = 0$ sinon.

La fonction appliquera la démarche suivante :

1. IC est initialisée par une matrice nulle.
2. Pour chaque pixel p de l'image IC. Si $IC_2[p] = 1$ alors :
 - (a) Créer une matrice booléenne v ayant les mêmes dimensions que IC_2 initialisée par des `False` partout.
 - (b) Calculer par la fonction `closure` les pixels non nuls de IC_2 connectés à p et mettre leurs contenus à 1 dans IC.

Solution:

```
def edge_map(ic_2):
    ic = np.zeros(ic_2.shape)
    h, w = ic_2.shape
    for x in range(h):
        for y in range(w):
            if ic_2[x,y] == 1:
                v = np.zeros(ic_2.shape, dtype = np.bool)
                r = closure(ic_2, v, (x,y))
                for x, y in r:
                    ic[x,y] = 1
    return ic
```

- 19) (1 point) Utiliser les fonctions précédentes pour définir une fonction `canny` permettant d'appliquer la démarche de Canny pour extraire les contours d'une image niveau de gris passée en entrée.

Solution:

```
def canny(Is, t_sup = 0.9 , t_inf = 0.2, n = 3 , sigma = 2.5):
    gauss = gaussian(n, sigma)
    I = gauss * Is
    g_rho, g_theta = polar_grad(I)
    g_dir = direction(g_theta)
    ic_1 = suppression(g_rho, g_dir)
    ic_2 = double_threshold(ic_1, t_sup, t_inf)
    return edge_map(ic_2)
```

Université de Carthage Institut Préparatoire aux Etudes d'ingénieur de Nabeul Département Mathématiques et Informatique		Année universitaire : 2017/2018
		Filières : M,P&T
		Niveau d'études : 2ème année
		Semestre : 2
		Nombre de pages : 4
		Date : 02/05/2018 Durée : 2h

Concours blanc

L'exercice 2 sera compté comme DS du deuxième semestre à traiter séparément sur un cahier à part

Exercice 1 : Programmation orientée objet (8pts)

On suppose créer un ensemble de classes qui permettent d'effectuer la gestion d'une bibliothèque. Pour ce faire, on vous demande de :

- 1) Ecrire une classe **Exemplaire** définie par deux attributs publics **num** et **dispo**, un constructeur et une méthode **aff** pour l'affichage. L'attribut **dispo** est de type booléen et indique si le livre est disponible ou pas.

Exemple : Pour e= ('AZ789EE-1', True) : on obtient l'affichage suivant :

N°Exemplaire :AZ789EE-1

Etat : Disponible

- 2) Ecrire une classe **Livre** définie par les attributs publics suivants :

- **isbn** : code du livre
- **titre** : titre du livre
- **auteur** : auteur du livre
- **nbex** : nombre d'exemplaires initialement nul

et **L** : un attribut privé de type liste initialement vide qui servira à stocker les exemplaires qui seront créés ultérieurement.

Prévoir un **constructeur** et les méthodes suivantes :

- **affLiv** : affiche les attributs isbn, titre et auteur du livre
- **affEx** : affiche les informations relatives aux exemplaires du livre
- **ajouterEx** : ajoute un exemplaire du livre à la liste des exemplaires.

NB : pour le livre l=('AZ78TT2', 'Programmer en Python', 'AB CB', ...), le numéro du premier exemplaire doit avoir l'allure suivante : 'AZ78TT2-1', celui du deuxième est 'AZ78TT2-2' et ainsi de suite...

- **emprunter** : retourne l'attribut **num** du premier exemplaire disponible en effectuant les mises à jour nécessaires et affiche le message 'livre non disponible' dans le cas échéant.
- **rendre** : permet de modifier les attributs de l'exemplaire numéro **num** afin qu'il devienne disponible de nouveau.

- 3) Ecrire une classe **Biblio** qui représente une bibliothèque de livres définie par un seul attribut privé **d** de type dictionnaire ayant la structure suivante :

- Pour chaque livre, on crée deux entrées :
 - La première utilise l'isbn du livre comme clé et son titre comme valeur
 - La deuxième c'est l'inverse : le titre représente la clé et l'isbn la valeur

Exemple : pour l=('AZ78TT','POO en Python', 'ARRR DFFG') , le dictionnaire stockera d={'AZ78TT' : 'POO en Python', 'POO en Python' : 'AZ78TT',...}

On vous demande de définir :

- Un attribut de classe (ou statique) appelé **nbliv**, sorte de compteur de livres initialement nul et qui sera mis à jour dans les méthodes en cas de besoin ;
- Un **constructeur** qui initialise le dictionnaire **d** au vide;
- Une méthode **ajoutLiv** qui ajoute un livre **liv** dans la bibliothèque s'il n'existe pas déjà ;
- Une méthode **rechLiv** : affiche l'**isbn** du livre si la recherche est basée sur le **titre** et affiche le **titre** si l'**isbn** est utilisé comme critère de recherche. Si le livre n'existe pas, un message d'erreur sera affiché. La valeur de l'isbn ou du titre est passée en paramètre dans une variable appelée **val**.
- Une méthode **suppLiv** : supprime un livre à partir de son **isbn** ou de son **titre**.
NB : on suppose que le livre existe dans la bibliothèque.
- Une méthode **compterLiv** qui affichera la valeur du compteur **nbliv**.

Exercice 2 : Bases de données (10pts)

On considère la base de données « base1 » constituée des trois relations suivantes :

- Acteur(numa,nom,spec,taille,poids)

Colonne	Description	Type
numa	Numéro de l'acteur	Entier
nom	Nom de l'acteur	Texte
spec	Spécialité de l'acteur	Texte
taille	Taille	Réel
poids	Poids	Réel

- Film(numf,titre,genre,annee,duree,budget,realisateur)

Colonne	Description	Type
numf	Numéro du film	Entier
titre	Titre du film	Texte
genre	Genre du film (comique,drame ...)	Texte
annee	Année de sortie	Entier
duree	Durée	Entier
budget	budget	Réel
realisateur	Réalisateur	Texte

- Jouer (#numf,#numa,desc,sal)

Colonne	Description	Type
numf	N° du film	Entier
numa	N° de l'acteur	Entier
desc	Description du rôle	Texte
sal	Salaire pour le rôle	Réel

- Les clés primaires sont soulignées et les clés étrangères sont précédées par #.
- Tous les attributs doivent être non vides

- 1) Donner les instructions **SQL** qui permettent de créer les trois relations de la base de données « **base1** » en respectant toutes les contraintes (de domaine, d'intégrité et référentielles).
- 2) Ecrire un script python qui :
 - Charge le module **sqlite3** ;
 - Etablit une connexion avec la base « **base1** »
 - Insère dans la table **Acteur** autant de lignes que l'utilisateur le souhaite
 - Et affiche le contenu de la table **Acteur**.
- 3) Répondre en algèbre relationnelle puis en **SQL** aux requêtes suivantes :
 - a) Afficher le titre des films qui ont une durée supérieure à 80 min et un budget ne dépassant pas les 100000 dinars ;
 - b) Afficher les rôles joués par l'actrice « Julia Roberts ».
- 4) Répondre en **SQL** uniquement aux questions suivantes :
 - a) Afficher l'intitulé du genre ainsi que le nombre de films correspondants ;
 - b) Afficher les genres qui ont plus que 20 films ;
 - c) Donner le titre et l'année du/de(s) film(s) le(s) plus long(s) ;
 - d) Calculer le total des salaires des acteurs qui ont joué dans « Taxi2 » ;
 - e) Afficher les noms des acteurs qui n'ont joué dans aucun film jusqu'à maintenant ;

Exercice 3 : Algèbre linéaire (7pts)

La factorisation Cholesky, consiste, pour une matrice carrée **A** d'ordre **n** symétrique définie positive, à déterminer une matrice triangulaire inférieure **L** telle que **A=LL'** (avec **L'** = transposé de **L**). Les relations suivantes permettent de déterminer les éléments de **L** à partir de **A** :

- Initialement tous les éléments de **L** sont nuls
- $L_{00} = \sqrt{A_{00}}$
- $L_{j0} = \frac{A_{0j}}{L_{00}}$ avec $j=1..n-1$
- $L_{ii} = \sqrt{A_{ii} - \sum_{k=0}^{i-1} L_{ik}^2}$ avec $i=1..n-1$
- $L_{ji} = \frac{(A_{ij} - \sum_{k=0}^{i-1} L_{ik} L_{jk})}{L_{ii}}$ avec $i=1..n-1$ et $j=i+1..n-1$

Le déterminant de **A** est égal alors au carré du produit des éléments de la diagonale principale de **L**.

- 1) Ecrire une fonction **DECOMP** qui calcule la matrice **L** à partir d'une matrice **A** d'ordre **n** en utilisant les formules précédentes.
- 2) Ecrire une fonction **DETER** qui retourne le déterminant d'une matrice **A** d'ordre **n** sans utilisation de la commande **det**.

L'inverse d'une matrice carrée **A** d'ordre **n** est donné par la formule suivante :

$$\mathbf{A}^{-1} = \mathbf{C}' / \det(\mathbf{A})$$

Notation:

C = matrice des cofacteurs dont les coefficients sont obtenus en utilisant la formule suivante :

$$C_{ij} = (-1)^{i+j} \det(A_{ij})$$

Avec : C_{ij} : L'élément de C à la ligne i et colonne j

A_{ij} : La matrice A privée de la ligne d'indice i et de la colonne d'indice j

$\det$: Le déterminant de la matrice

C' : transposé de C

- 3) Ecrire une fonction **MINOR** qui supprime la ligne d'indice i et la colonne d'indice j d'une matrice A d'ordre n .
- 4) Ecrire une fonction **COFACT** qui calcule et retourne la matrice des cofacteurs d'une matrice A d'ordre n .
- 5) Ecrire une fonction **INVERSE** qui retourne l'inverse d'une matrice A d'ordre n .

Exercice 4 : Traitement d'image (5pts)

Le filtre de Prewitt utilise la matrice $A=[[-1,0,-1],[-1,0,1],[-1,0,1]]$ afin de définir le contour dans une image en niveaux de gris. Le principe consiste à calculer le produit de convolution pour chaque pixel de l'image originale par la matrice A comme l'illustre le schéma suivant :

p1	p2	p3		
p4	p5	p6		
p7	p8	p9		
.....				

-1	0	1
2	1	-2
0	2	1

				
		S		
.....				

Si on considère le pixel **p5** de l'image à gauche, le produit de convolution par la matrice $A=[[-1,0,-1],[2,1,-2],[0,2,1]]$ est égal à :

$S=-1*p1+0*p2+1*p3+2*p4+1*p5+(-2)*p6+0*p7+2*p8+1*p9$ et qui sera enregistrée dans la nouvelle image à droite à la même position que le pixel p5.

- 1) Ecrire une fonction **Convolution** qui étant donné neufs pixels **p1, p2, p3, p4, p5, p6, p7, p8** et **p9** et une liste de listes **A** représentant un matrice **3x3**, calcule le produit de convolution du pixel **p5** comme le montre le schéma précédent.
- 2) Ecrire une fonction **AppFiltreConv** qui prend en paramètre un tableau **T** à **nc** colonnes et **nl** lignes stockant les pixels d'une image en niveaux de gris, un filtre **A**, et une image résultat **img2**, et qui applique la convolution sur tous les pixels de **T**.
NB : Attention aux débordements
- 3) Ecrire un script python qui :
 - Charge la classe **Image** du module **PIL** ;
 - Crée un objet **img** stockant l'image enregistrée dans 'c:\python32\image.jpg' ;
 - Convertit en une seule commande **img** en une image en niveaux de gris **img1** ;
 - Applique le filtre de Prewitt sur **img1** afin de créer une image **img2** contenant les contours de **img** ;
 - Et affiche **img2**.

Alternative du corrigé du concours blanc d'informatique @IPEIN AU 2017/2018

Exercice 1 : 8 pts

1)	1 pts = 0.5 + 0.5	<pre> class Exempleaire: def __init__(self,num): self.num=num self.dispo=True def aff(self): print('N°Exempleaire:{}'.format(self.num)) if self.dispo:print('Etat:Disponible') else:print('Etat:Non disponible') </pre>
2)	3,5 pts= 0.25 0.25 0.5 1 0.75 0.75	<pre> class Livre: def __init__(self,isbn,titre,auteur): self.isbn,self.titre,self.auteur=isbn,titre,auteur self.nbex=0 self.__L=[] def affLiv(self): print('ISBN:{}\nTitre:{}\nAuteur:{}\n'. format(self.isbn,self.titre,self.auteur)) def affEx(self): for i in self.__L:i.aff() def ajouterEx(self): self.nbex+=1 num=self.isbn+'-'+str(self.nbex) self.__L.append(Exempleaire(num)) def emprunter(self): for i in self.__L: if i.dispo: i.dispo=False return i.num print('Non disponible') def rendre(self,num): postiret=num.index('-') pos=int(num[postiret+1:]) self.__L[pos-1].dispo=True </pre>

3)	3,5 = 0.25 0.25 1 0.75 1 0.25	<pre> class Biblio: nbLiv=0 def __init__(self): self.__d={} def ajouterLiv(self,liv): if liv.isbn not in self.__d and liv.titre not in self.__d: self.__d[liv.isbn]=liv.titre self.__d[liv.titre]=liv.isbn Biblio.nbLiv+=1 def rechLiv(self,val): if val in self.__d: print(self.__d[val]) else:print('Non existant') def suppLiv(self,val): val1=self.__d[val] del self.__d[val] del self.__d[val1] Biblio.nbLiv-=1 def compterLiv():print(Biblio.nbLiv) </pre>
----	---	--

Exercice 3 : 7 pts

1)	2 pts	<pre> def DECOMP(A,n): from numpy import zeros L=zeros((n,n)) from math import sqrt L[0,0]=sqrt(A[0,0]) for j in range(1,n):L[j,0]=A[0,j]/L[0,0] for i in range(1,n): s=0 for k in range(i):s+=L[i,k]**2 L[i,i]=sqrt(A[i,i]-s) for j in range(i+1,n): s1=0 for k in range(i):s1+=L[i,k]*L[j,k] L[j,i]=(A[i,j]-s1)/L[i,i] return L </pre>
2)	1 pt	<pre> def DETER(A,n): L=DECOMP(A,n) p=1 for i in range(n): p*=L[i,i] return p**2 </pre>
3)	1 pts	<pre> def MINOR(A,i,j): from numpy import delete from copy import copy A1=copy(A) return delete(delete(A1,i,0),j,1) </pre>

4)	1,5 pts	<pre>def COFACT(A,n): from numpy import array from numpy.linalg import det C=array([[None]*n]*n) for i in range(n): for j in range(n): C[i,j]=-1**(i+j)*det(MINOR(A,i,j)) return C</pre>
5)	1,5 pts	<pre>def INVERSE(A,n): C=COFACT(A,n) from numpy import transpose return 1/det(A)*transpose(C)</pre>

Exercice 4 : 5 pts

1)	1 pt	<pre>def Convolution(p1,p2,p3,p4,p5,p6,p7,p8,p9,A): return int((A[0][0]*p1+A[0][1]*p2+A[0][2]*p3+ A[1][0]*p4+A[1][1]*p5+A[1][2]*p6+ A[2][0]*p7+A[2][1]*p8+A[2][2]*p9)/9)</pre>
2)	2 pts	<pre>def AppFiltreConv(T,nc,nl,A,img2): for c in range(1,nc-1): for l in range(1,nl-1): p1=T[c-1,l-1] p2=T[c,l-1] p3=T[c+1,l-1] p4=T[c-1,l] p5=T[c,l] p6=T[c+1,l] p7=T[c-1,l+1] p8=T[c,l+1] p9=T[c+1,l+1] pixel=Convolution(p1,p2,p3,p4,p5,p6,p7,p8,p9,A) img2.putpixel((c,l),pixel)</pre>
3)	2pts = 8*0. 25	<pre>from PIL import Image nomf='c:\python32\image.jpg' img=Image.open(nomf) nc,nl=img.size img1=img.convert('L') img2=Image.new(img.mode,img.size) T=img1.load() AppFiltreConv(T,nc,nl,[[[-1,0,1],[-1,0,1],[-1,0,1]],img2) img2.show()</pre>

Université de Carthage Institut Préparatoire aux Etudes d'ingénieur de Nabeul Département mathématiques		Année universitaire : 2018/2019	
		Filières : SM, ST & SP	
		Niveau d'études : 2ème année	
		Semestre : 1	
		Nombre de pages : 3	
		Date : 15/12/2018	Durée : 2h

Examen du 1^{er} semestre

Exercice 1 : (5,5 pts)

1) Ecrire une classe **Personne** définie par les attributs publics suivants :

- **cin** : Numéro de la carte d'identité ;
- **nom** : Le nom de la personne
- **pre** : Le prénom de la personne
- **adr** : L'adresse
- **tel** : Le numéro du téléphone

Prévoir un **constructeur** et la fonction spéciale **__repr__** pour l'affichage.

Exemple :

Pour une personne p = (cin=78414111, nom='Ben Salah',pre='Med',adr='BP 1000, Tunis',tel=25747789) , on obtiendra :

NCIN : 78414111

Nom & Prénom: Ben Salah Med

Contact : BP 1000, Tunis – Tél : 25747789

2) On souhaite développer une classe **Etudiant** qui représente un étudiant à l'université. Chaque étudiant peut choisir d'étudier quelques matières. Les matières proposées ainsi que leurs coefficients respectifs sont résumés dans le tableau suivant :

Code de la matière	Coefficient
math	6
info	3
chim	4
phy	4
fr	2
ang	2

Ecrire une classe **Etudiant** qui hérite de la classe **Personne** et qui y rajoute :

- Un attribut public **numcarte** représentant le n° de la carte d'étudiant de type **str** ;
- Un attribut privé appelé **Notes** de type dictionnaire initialement vide avec comme **clés** les **codes des matières** et comme **valeurs les notes** correspondantes;
- Deux attributs de classe, le premier nommé **mat**, le deuxième **coef**, sorte de tuples stockant respectivement les codes des matières et leurs coefficients respectifs.

On vous demande de définir :

- Un **constructeur** ;
- La fonction spéciale **__repr__** (vous pouvez utiliser le résultat fourni par la même fonction au niveau de la classe mère) afin d'afficher tous les attributs y compris les matières et les notes obtenues ;

- c) Une méthode **ajoutNote** qui saisit le code d'une matière et la note correspondante en effectuant les contrôles nécessaires, puis l'ajoute dans le dictionnaire **Notes** ;
- d) Une méthode **modifNote** qui modifie la note d'une matière qui figure déjà dans le dictionnaire des notes ;
- e) Une méthode **calculMoy** qui retourne la moyenne des notes;
- f) Une méthode **memesCours** qui teste si deux étudiants ont choisi d'étudier les mêmes cours ;
- g) Et une méthode **comparer** qui compare la moyenne de deux étudiants qui ont eu les mêmes cours. La méthode retourne **-1** quand le premier étudiant devance le deuxième, **0** quand les deux moyennes sont égales et **1** dans le cas échéant.

Exercice 2 : (8 pts)

IL EST INTERDIT D'UTILISER LES COMMANDES DE LA CLASSE **str**

Les commandes **ord** et **chr** sont permises

On souhaite développer une classe nommée **chaine** qui permettra de créer et de manipuler des chaînes de caractères. Un objet de type **chaine** possède un seul attribut **L** privé, de type liste de caractères. Exemple : La liste ['p','y','t','h','o','n'] définit la chaîne « python ».

Travail demandé :

- 1) Ecrire le constructeur de la classe **chaine** qui initialise **L** à une liste vide.
- 2) Définir les méthodes suivantes :
 - **taille** : retourne le nombre de caractères d'une chaîne ;
 - **saisie** : saisit les éléments d'une chaîne caractère par caractère. A la fin de la saisie de chaque caractère, on propose à l'utilisateur de répondre pour 'o' pour passer à la saisie du caractère suivant, sinon on arrête le traitement.
NB : La table Ascii contient 128 caractères avec des codes variant de 0 à 127.
 - **alpha** : teste si une chaîne est composée uniquement de lettres ;
 - **num** : teste si une chaîne est composée uniquement de chiffres ;
 - **espace** : teste si la chaîne est un espace ;
 - **majus** : convertit les lettres minuscules d'une chaîne en lettres majuscules (Les caractères non alphabétiques resteront inchangés);
 - **minuscule** : convertit les lettres majuscules d'une chaîne en lettres minuscules (Les caractères non alphabétiques resteront inchangés ;
 - **affich** : affiche les caractères d'une chaîne après leur concaténation sur une ligne en les encadrant par des chevrons comme le montre l'exemple au début de l'énoncé ;
 - **pos** : retourne la position de la première occurrence d'un caractère **c** dans une chaîne (Si **c** n'existe pas dans la chaîne, on ne retourne rien) ;
 - **occ** : calcule et retourne le nombre d'occurrences d'un caractère **c** dans une chaîne
 - **remplace** : remplace un caractère **c** par un autre appelé **c1** dans une chaîne ;
 - **decoupe** : découpe une chaîne en utilisant un caractère de découpage noté **c**, le résultat est une liste de listes
Exemple : pour la chaîne « salut soir des » et **c='s'** : on obtient :
[[''], ['a', 'l', 'u', 't', ''], ['o', 'i', 'r', ''], ['d', 'e', 's']].
 - **sousChaine** : retourne un objet de type chaîne, composé des lettres qui commencent à la position **deb** et se terminent à la position **fin**.
- 3) Proposer une solution pour pouvoir appliquer :
 - L'opérateur + à deux objets de type **chaine** afin de les concaténer ;
 - L'opérateur * pour répéter un objet de type chaîne **n** fois ;

- Et l'opérateur == pour tester l'égalité de deux objets de type chaînes.

Exercice 3 : (6,5 pts)

- 1) Les listes de listes sont utiles pour représenter une matrice de distances entre **n** villes numérotées **0, 1, 2, ..., n-1** comme le montre l'exemple suivant :

Exemple : on considère la liste de listes **D** suivante (avec n=6) :

$D = [[0,1,2,0,0,0], [1,0,3,2,0,0], [2,3,0,1,2,0], [0,2,1,0,4,2], [0,0,2,4,0,3], [0,0,0,2,3,0]]$:

- $D[0][1]=1$: Distance entre la ville n°0 et la ville n°1 égale à 1 ;
- $D[4][5]=3$: Distance entre la ville n°4 et la ville n°5 égale à 3 ;
- $D[i][i]=0$ quelque soit $i=0..5$ (une ville n'est pas reliée à elle-même)

On souhaite créer une classe **Villes** qui permet de représenter les distances entre **n** villes. Un objet de type **Villes** est caractérisé par les attributs publics suivants :

- **n** : nombre de villes
- **D** : une liste de listes stockant les distances entre les **n** villes

On vous demande de définir :

- a) Un **constructeur** ayant comme paramètre une liste de listes appelée **D** et qui donne des valeurs de départ aux attributs **n** et **D** en respectant la définition déjà donnée (utiliser la commande **copy**) ;
 - b) Une méthode **voisins** qui retourne une liste contenant les numéros des villes auxquelles une ville numéro **num** est reliée directement (distance non nulle).
Exemple : pour la ville numéro 3, la liste des voisins = [1, 2, 4, 5]
 - c) Une méthode **degreMax** qui retourne le(s) numéro(s) de(s) ville(s) ayant le plus de voisins ;
 - d) Une méthode **extraire** qui retourne une liste de tuples chacun de la forme (**ville1,ville2**) représentant un chemin entre deux villes en considérant chaque couple de villes une seule fois.
- 2) Un objet de type **Villes** est dit connexe si chaque ville est accessible via une route à partir de n'importe quelle autre ville. Pour ce faire, on utilisera un algorithme basé sur les files d'attente.
 - a) Ecrire une classe **File** ayant un seul attribut public appelé **L** de type liste initialement vide, un **constructeur** et les trois méthodes **enfiler**, **défiler** et **vide**.
 - b) Définir la méthode **connexe** qui teste si un objet de type **Villes** est connexe. La méthode possède un seul paramètre appelé **VDEP** qui représente la ville de départ à partir de laquelle on commencera la vérification et elle est basée sur le pseudo algorithme suivant :

- Créer un objet **F** de type File et un ensemble **P** initialement vide ;
- Ajouter **VDEP** dans **P** et l'enfiler dans **F** ;
- On répète les étapes suivantes :
 - On défile la file afin d'obtenir le numéro d'une ville **v** ;
 - Pour chaque voisin de **v** : s'il n'est pas encore dans **P** alors on l'ajoute et on l'enfile dans la file ;
 - On s'arrête quand la file devienne vide.

La méthode retourne la valeur **True** si l'ensemble **P** contient les numéros des **n** villes à la fin de l'exécution et **False** dans le cas contraire.

Alternative du corrigé de l'examen d'informatique du 1^{er} semestre - 2^{èmes} années (Barème sur 40)

Exercice 1 : (11 pts)

1)	<pre>>>> class Personne: def __init__(self,cin,nom,pre,adr,tel): self.cin=cin self.nom=nom self.pre=pre self.adr=adr self.tel=tel def __repr__(self): return 'NCIN:{}\nNom& Prénom:{}\nAdresse:{}\nTél:{}' .format(self.cin,self.nom+' '+self.pre,self.adr,self.tel)</pre>	1,5 pts
2)		
a)	<pre>class Etudiant(Personne): mat=('info','chim','math','phy','fr','ang') coef=(3,4,6,4,2,2) def __init__(self,cin,nom,pre,adr,tel,numcarte): Personne.__init__(self,cin,nom,pre,adr,tel) self.numcarte=numcarte self.__Notes={} ...</pre>	1,5 pts
b)	<pre>def ajoutNote(self): print('choisir parmi les matieres suivantes:') for i in Etudiant.mat: print(i) print() while 1: m=input('donner une matiere') if m in Etudiant.mat and m not in self.__Notes: break while 1: try:note=float(input('donner une note')) except:print('error') else: if 0<=note<=20: break self.__Notes[m]=note</pre>	2 pts
c)	<pre>def modifNote(self,m,note): if m in self.__Notes:self.__Notes[m]=note ...</pre>	0,5 pt
d)	<pre>def __repr__(self): s=Personne.__repr__(self) s+='N°Carte:{}\n' .format(self.numcarte) s+='Liste des notes:' for m,note in self.__Notes.items(): s+='{}:{}\n' .format(m,note) return s</pre>	1,5 pts

e)	<pre>def calculMoy(self): sn=0 sc=0 for m,note in self.__Notes.items(): coeff=Etudiant.coef[Etudiant.mat.index(m)] sc+=coeff sn+=coeff*note return sn/sc</pre>	2 pts
f)	<pre>def memesCours(self,e): return set(self.__Notes.keys())==set(e.__Notes.keys())</pre>	0.5 pt
g)	<pre>def comparer(self,e): if self.memesCours(e)==False:return moy1=self.calculMoy() moy2=e.calculMoy() if moy1==moy2:return 0 elif moy1<moy2:return -1 else:return 1</pre>	1,5 pts

Exercice 2 : (16 pts)

NB : certaines questions supportent deux versions de réponses dépendantes de la connaissance ou non des étudiants des codes ascii des lettres a, A, z et Z.

1)	<pre>class chaine: def __init__(self): self.__L=[]</pre>	0.5 pt
2)		
•	<pre>def taille(self): return len(self.__L)</pre>	0.5 pt
•	<pre>def saisie(self): i=0 while 1: while 1: c=input('donner s[{}].format(i)) if c in [chr(i) for i in range(128)]: self.__L.append(c) i+=1 break else: print('tapez un caractere') rep=input('tapez o pour continuer') if rep!='o':break</pre>	2 pts
•	<pre>Version 1 : def alpha(self): # solution 1 LMIN=[chr(i) for i in range(ord('a'),ord('z')+1)] LMAJ=[chr(i) for i in range(ord('A'),ord('Z')+1)] L=LMIN+LMAJ for i in self.__L: if i not in L:return False return True</pre>	1 pt

	Version 2 : <pre>def alpha2(self): # solution 2 for i in self.__L: if ord(i) not in range(65,91) and ord(i) not in range(97,123): return False return True</pre>	
•	<pre>def num(self): for i in self.__L: if i not in ['0','1','2','3','4','5','6','7','8','9']:return False return True</pre>	1 pt
•	<pre>def espace(self):return len(self.__L)==1 and self.__L[0]==' '</pre>	1 pt
•	Version 1 : <pre>def majus(self): LMIN=[chr(i) for i in range(ord('a'),ord('z')+1)] LMAJ=[chr(i) for i in range(ord('A'),ord('Z')+1)] d={LMIN[i]:LMAJ[i] for i in range(26)} for i in range(self.taille()): if self.__L[i] in d: self.__L[i]=d[self.__L[i]]</pre> Version 2 : <pre>def majus2(self): for i in range(len(self.__L)): codec=ord(self.__L[i]) if codec in range(65,91) or codec not in range(97,123): continue self.__L[i]=chr(codec-32)</pre>	1 pt
•	Version 1 <pre>def minuscule(self): LMIN=[chr(i) for i in range(ord('a'),ord('z')+1)] LMAJ=[chr(i) for i in range(ord('A'),ord('Z')+1)] d={LMAJ[i]:LMIN[i] for i in range(26)} for i in range(self.taille()): if self.__L[i] in d: self.__L[i]=d[self.__L[i]]</pre> Version 2 <pre>def minuscule2(self): for i in range(len(self.__L)): codec=ord(self.__L[i]) if codec in range(97,123) or codec not in range(65,91): continue self.__L[i]=chr(codec+32)</pre>	1 pt

•	<pre>def affich(self): print(' ',end='') for i in self.__L: print(i,end='') print(' ',end='')</pre>	1 pt
•	<pre>def pos(self,c): for i in range(self.taille()): if self.__L[i]==c:return i</pre>	1 pt
•	<pre>def occ(self,c): return self.__L.count(c)</pre>	0.5 pt
•	<pre>def replace(self,c,c1): for i in range(self.taille()): if self.__L[i]==c: self.__L[i]=c1</pre>	1 pt
•	<pre>def decoupe(self,c): i=0 a=[] while i<len(self.__L): j=i while j<len(self.__L) and self.__L[j]!=c: j+=1 a.append(self.__L[i:j]) i=j+1 return a</pre>	2 pts
•	<pre>def sousChaine(self,deb,fin): sch=chaine() sch.__L=self.__L[deb:fin+1] return sch</pre>	1 pt
3)	<pre>def __add__(self,ch): r=chaine() r.__L=self.__L+ch.__L return r def __mul__(self,n): r=chaine() r.__L=self.__L*n return r def __eq__(self,ch): return self.__L==ch.__L</pre>	1.5pts

Exercice 3: (13 pts)

1)		
a)	<pre>class Villes: def __init__(self,D): from copy import copy self.D=copy(D) self.n=len(D)</pre>	1.5 pts

b)	<pre>def voisins(self,num): a=[] for i in range(self.n): if self.D[num][i]!=0: a.append(i) return a</pre>	2 pts
	<pre>def degMax(self): L=[len(self.voisins(i)) for i in range(self.n)] return [i for i in range(len(L)) if L[i]==max(L)]</pre>	2 pts
	<pre>def extraire(self): a=[] for i in range(self.n): for j in range(i+1): if self.D[i][j]!=0:a.append((i,j)) return a</pre>	2 pts
2)		
a)	<pre>class File: def __init__(self): self.L=[] def vide(self):return self.L==[] def enfiler(self,x):self.L.insert(0,x) def defiler(self): if self.vide()==False:return self.L.pop()</pre>	2 pts
b)	<pre>def connexe(self,vdep): f=File() P={vdep} f.enfiler(vdep) while f.vide()==False: x=f.defiler() for v in self.voisins(x): if v not in P: P.add(v) f.enfiler(v) return P==set(range(self.n))</pre>	3.5 pts

AVERTISSEMENT

- L'épreuve est composée de 4 exercices, totalement indépendants.
- Vous pouvez toujours admettre le résultat des questions non faites pour faire les questions suivantes.

Exercice 1 : (3pts)

1. Citez deux avantages de la programmation orientée objet par rapport à la programmation procédurale. [1]
2. Citez un exemple d'un problème scientifique simple où sa résolution nécessite l'utilisation d'une structure de données avancée de type **Pile** ou bien de type **File** afin d'enregistrer vos données et vos résultats. Justifiez votre choix. [1]
3. Théoriquement, une structure de données avancée de type **Pile** peut avoir une capacité de stockage infinie. Selon vous, pratiquement, quelle est la contrainte majeure de ne pas avoir une pile à capacité infinie ? [1]

Exercice 2 : (3pts)

1. Qu'affichera ce programme ? Justifiez votre choix. [1]

```
class parent:
    def __init__(self, param):
        self.v1 = param

class enfant(parent):
    def __init__(self, param):
        self.v2 = param

obj = enfant(11)
print(obj.v1 + " " + obj.v2)
```

- a) None None
- b) None 11
- c) 11 None
- d) 11 11
- e) AttributeError

2. Qu'affichera ce programme ? Justifiez votre choix.

[1]

```
class Compte:
    def __init__(self, id):
        self.id = id
        id = 100

cpt = Compte(123)
print(cpt.id)
```

- a) SyntaxError, ce programme ne fonctionnera pas
- b) None
- c) 123
- d) 100
- e) Aucune de ces réponses

3. Quelles sont les réponses correctes à propos de l'extrait du code donné ?

[1]

```
class A:
    def __init__(self, i = 0):
        self.i = i

class B(A):
    def __init__(self, j = 0):
        self.j = j

def main():
    b = B()
    print(b.i)
    print(b.j)

main()
```

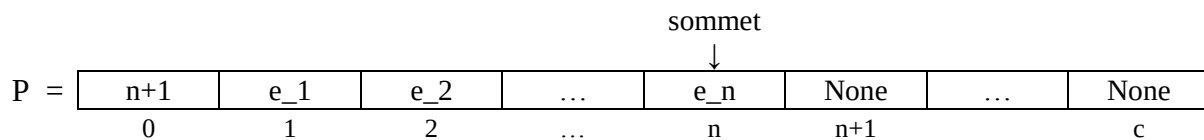
- a) La classe B hérite de A, mais le champ de données "i" dans A n'est pas hérité.
- b) La classe B hérite de A, héritant ainsi automatiquement tous les attributs de données de A.
- c) Lorsque vous créez un objet de B, vous devez passer un argument tel que B(5).
- d) Le champ de données "j" n'est pas accessible par l'objet b.

Exercice 3 : (6pts)

Dans le cadre de cet exercice, on veut réaliser, à l'aide d'une liste, une pile à capacité finie permettant de stocker au plus **c** éléments:

- La première case (de rang **0**) contient l'indice du prochain élément à insérer dans la pile.
- Lorsque **P [0]** égale à **1**, la pile est considérée comme vide.
- Les **n** cases suivantes, d'indices de **1** à **n**, contiennent les éléments déjà insérés dans la pile.
- Le dernier élément inséré, d'indices de **n**, est appelé sommet de la pile.
- Les cases restantes, d'indices de **n+1** à **c**, n'appartiennent pas à la pile.
- À chaque fois qu'on insère un élément, on augmente **P [0]** d'une unité.
- Lorsque **P [0]** égale à **c+1**, la pile est considérée comme pleine.

Exemple : Une pile P, non pleine, contenant n éléments.



Travail à faire :

1. Ecrire une classe Python nommée « **MaPile** » implémentant une pile d'éléments. Définir le **constructeur** de cette classe qui initialisera une pile vide à une capacité maximale de **c** éléments. [1]
2. Définir une méthode **est_vide** permettant de retourner True si la pile est vide, sinon elle retourne False. [0.5]
3. Définir une méthode **est_pleine** permettant de retourner True si la pile est pleine, sinon elle retourne False. [0.5]
4. Définir une méthode **sommets** permettant de retourner la valeur du sommets de la pile. [0.5]
5. Définir une méthode **empiler** permettant d'ajouter un élément **e** à la pile. Cette méthode déclenche une exception de type IndexError si la pile est pleine. [1.5]
6. Définir une méthode **dépiler** permettant de retourner le sommets de la pile et de le mettre à jour à la bonne position. Cette méthode déclenche une exception de type IndexError si la pile est vide. [1.5]
7. Définir une méthode **__str__** permettant d'imprimer les éléments de la pile. [0.5]

Exemples d'utilisations de la classe MaPile :

```
>>> pile1 = MaPile(7)
>>> print(pile1)
[1, None, None, None, None, None, None, None]
>>> pile1.empiler('a')
>>> print(pile1)
[2, 'a', None, None, None, None, None, None]
>>> pile1.empiler('b')
>>> print(pile1)
[3, 'a', 'b', None, None, None, None, None]
>>> x = pile1.depiler()
>>> print(x)
b
>>> print(pile1)
[2, 'a', 'b', None, None, None, None, None]
```

Exercice 4 : (8 pts)

Programmons une classe **Vect2d** dont les objets modéliseront des vecteurs à deux composantes réelles dans un repère orthonormé ($O ; I ; J$), gradué avec la même unité ($OI = OJ = 1$ unité). Lorsque le mathématicien dit : « Soit le vecteur $\vec{u}(3,-2)$ », le programmeur Python dira : « $u = \text{Vect2d}(3,-2)$ ».

1. Programmez le *constructeur*, chargé d'initialiser les attributs x et y de l'instance courante, qui se nomme *self*. Par défaut, x et y seront mis à 0. [1]
2. Si je veux additionner deux vecteurs, j'ai deux solutions. Ou bien je programme une fonction mathématique **add(v1,v2)** à l'extérieur de la classe. Ou bien, dans une optique de pure programmation par objets, je programme **add** comme méthode d'instance à l'intérieur de la classe. Adoptez cette seconde solution, pour laquelle le résultat de l'addition avec la méthode **add** sera un nouveau vecteur. [1]
3. Programmez une méthode **mul_ext** réalisant la multiplication $k\vec{u}$ d'un réel k par un vecteur $\vec{u}$. Le résultat sera un nouveau vecteur. [1]
4. Programmez une méthode **zoom** permettant à un vecteur de se multiplier par k et d'être ainsi définitivement modifié ! Faites bien la différence avec **mul_ext**. [1]
5. Programmez une méthode **prodscal** demandant à un vecteur de retourner son produit scalaire avec un autre vecteur. [1]
6. En déduire une méthode **norme** demandant à un vecteur de retourner sa longueur. [0.5]
7. Programmez à l'extérieur de la classe une fonction **add(v1,v2)** retournant la somme vectorielle $\vec{v}_1 + \vec{v}_2$. [1]

Remarque : En principe il est impossible d'écrire $u + v$ si u et v sont deux vecteurs. Mais en Python, des méthodes spéciales sont cachées sous les opérateurs. Si votre méthode **add** se nomme **__add__**, cela devient possible !

8. Afin de tester votre classe **Vect2d** : [1.5]
 - a. Créez deux vecteurs $\vec{u}(3,-2)$ et $\vec{v}(4,1)$
 - b. Créez un vecteur $\vec{w} = \vec{u} + \vec{v}$
 - c. Vérifiez que $(\vec{u} + \vec{w}) \cdot \vec{v} = \vec{u} \cdot \vec{v} + \vec{w} \cdot \vec{v}$
 - d. Vérifiez que $(5\vec{u}) \cdot \vec{v} = 5(\vec{u} \cdot \vec{v})$
 - e. Affichez la valeur de l'angle α formé entre les deux vecteurs $\vec{u}$ et $\vec{v}$.
Rappelons que :
 $\vec{u} \cdot \vec{v} = \|\vec{u}\| \cdot \|\vec{v}\| \cdot \cos(\alpha)$
En python la fonction **acos(x)** du module **math** retourne **cos⁻¹(x)**, l'arc cosinus de x , en radians.
Exemple : si $\cos(x) = 1.0$ alors $x = \text{acos}(1.0) = 0.0$

... Fin de l'épreuve ...

Correction

Filières : SM, SP, ST

Durée : 2h

Exercice 1 : (3pts) Réponse

1. Parmi les avantages de la programmation orientée objet par rapport à la programmation procédurale : [1]
 - Réutilisation du code ; écrire une fois, utiliser plusieurs fois.
 - Permet la création des nouveaux types.
 - Permet d'étendre les fonctionnalités d'un programme grâce à l'héritage entre les classes
 - Code plus lisible, plus claire, plus modulaire.
 - Encapsulation :
 - Une fois qu'un objet est créé, la connaissance de son implémentation n'est pas nécessaire pour son utilisation. Dans les anciens programmes, les codeurs devaient comprendre les détails d'un morceau de code avant de l'utiliser
 - Protection des données grâce aux modificateurs : publiques, privés et protégés. Les objets ont la capacité de cacher certaines parties d'eux-mêmes aux programmeurs. Cela empêche les programmeurs de falsifier des valeurs qu'ils ne devraient pas.
2. [1]
 - **Problème 1 :** Enregistrement des pages web visités par un navigateur web dans une pile. Lorsque l'internaute visite des pages web lors de son navigation, elles seront enregistrées (empilées) dans l'ordre de visite chronologique. Si l'internaute veut réafficher une page déjà visitée, il doit cliquer plusieurs fois sur le bouton précédent (dépiler) jusqu'à ce qu'il arrive à la page voulue.
 - **Problème 2 :** Enregistrement des résultats des appels récurifs dans une pile d'exécution lors de calcul du factoriel d'un entier n quelconque.

1 * Factoriel (0)
...
n-1 * Factoriel (n-2)
n * Factoriel (n-1)
Factoriel (n)

- **Problème 3** : Utiliser une file d'attente pour enregistrer temporairement des documents en attente d'impression.

Fichier1.doc	Exercice1.txt	...	Test.pdf
En cours d'impression	fichier en attente d'impression		Dernier document ajouté
File d'attente			

- **Autres Problèmes ...**

3. Pratiquement, la contrainte majeure de ne pas avoir une pile à capacité infinie est **la taille limitée de la mémoire** de la machine sur laquelle on implémente notre pile.

[1]

Exercice 2 : (3pts)

Question	Réponse	Justification	
1	e) AttributeError	L'instance enfant n'a aucun attribut 'v1'. self.v1 n'a jamais été créé en tant que variable puisque le parent. __init__ n'était pas explicitement appelé.	[1]
2	c) 123	L'instanciation de classe appelle automatiquement la méthode __init__ et transmet 123 en tant que paramètre. 123 est affecté à l'attribut de données de l'objet appelé id. La valeur 100 n'est pas conservée dans l'objet car elle n'est pas affectée à un attribut de données de la classe / de l'objet.	[1]
3	a) La classe B hérite de A, mais le champ de données "i" dans A n'est pas hérité.		[1]

Exercice 3 : (6pts)

Question	Réponse	
1)	<pre>class MaPile () : def __init__(self, c = 0) : self.pile = [None] * (c+1) self.pile[0] = 1</pre>	[1]
2)	<pre>def est_vide (self) : return self.pile[0] == 1</pre>	[0.5]
3)	<pre>def est_pleine (self) : return self.pile[0] == len(self.pile)</pre>	[0.5]

4)	<pre>def sommet (self) : if not self.est_vide(): i = self.pile[0]-1 return self.pile[i]</pre>	[0.5]
5)	<pre>def empiler(self, e) : if not self.est_pleine(): i = self.pile[0] self.pile[i] = e self.pile[0] += 1 else: print("Débordement de pile, arrêt du programme") raise(IndexError)</pre>	[1.5]
6)	<pre>def depiler(self) : if self.est_vide(): print("Débordement négatif de pile : arrêt du programme") raise(IndexError) else: self.pile[0] -= 1 i = self.pile[0] return self.pile[i]</pre>	[1.5]
7)	<pre>def __str__(self): #cette method doit retourner une chaine de caractères ch = str(self.pile) return ch</pre>	[0.5]

Exercice 4 : (8 pts)

Question	Réponse	
1)	<pre>class Vect2d : def __init__(self, x=0, y=0): self.x = x self.y = y</pre>	[1]
2)	<pre>def add(self, autre): v = Vect2d() v.x = self.x + autre.x v.y = self.y + autre.y return v</pre>	[1]
3)	<pre>def mul_ext(self, k): v = Vect2d() v.x = self.x * k v.y = self.y * k return v</pre>	[1]
4)	<pre>def zoom (self, k): self.x *= k self.y *= k</pre>	[1]

5)	<pre>def prodscal (self,autre): x = self.x * autre.x y = self.y * autre.y return x+y</pre>		[1]
6)	<pre>def norme (self): from math import sqrt return sqrt(self.x **2 + self.y **2)</pre>		[0.5]
7)	<pre>#En dehors de la classe def add(v1,v2): v = Vect2d() v.x = v1.x + v2.x v.y = v1.y + v2.y return v</pre>		[1]
8)	<pre>#Programme principal #a) u = Vect2d(3,-2) v = Vect2d(4,1)</pre>	[0.25]	[1.5]
	<pre>#b) w = u.add(v) print(w.x, w.y)</pre>	[0.25]	
	<pre>#c) ps1 = w.add(u).prodscal(v) ps2 = u.prodscal(v) + w.prodscal(v) print(ps1 == ps2)</pre>	[0.25]	
	<pre>#d) from copy import copy u1 = copy(u) u1.zoom(5) #5u ps1 = u1.prodscal(v) ps2 = u.prodscal(v) ps2 *=5 print(ps1 == ps2)</pre>	[0.25]	
	<pre>#e) a = u.prodscal(v) /(u.norme() * v.norme()) from math import acos alpha = acos(a) print(alpha)</pre>	[0.5]	

... Fin de correction ...

Université de Carthage Institut Préparatoire aux Etudes d'Ingénieur de Nabeul Département de Mathématique et de l'Informatique	 المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2019/2020 Filière : SM/SP/ST Niveau d'étude : 2^{ème} année Semestre : 2 / Date : 20/02/2020 Nombre de pages : 2 Durée : 1h30
--	---	--

Devoir Surveillé de : Informatique

Afin d'éviter les collisions entre avions, les altitudes de vol en croisière sont normalisées. Dans la majorité des pays, les avions volent à une altitude multiple de 1000 pieds (un pied vaut 30,48 cm) au dessus de la surface isobare à 1013,25 hPa (C'est le niveau de la mer « niveau de vol zéro »). L'espace aérien est ainsi découpé en tranches horizontales appelées niveaux de vol et désignées par les lettres « FL » (Flight Level) suivies de l'altitude en centaines de pieds : « FL310 » désigne une altitude de croisière de 31000 pieds au-dessus de la surface isobare de référence.

OACA (Office de l'Aviation Civile et des Aéroports) est l'organisme tunisien chargé de la navigation aérienne, il gère des centaines de vol par jour. Toute compagnie qui souhaite faire traverser le ciel tunisien à un de ses avions doit soumettre à cet organisme un plan de vol comprenant un certain nombre d'informations : trajet, heure de départ, niveau de vol souhaité, etc. Muni de ces informations, OACA peut prévoir les secteurs aériens qui vont être surchargés et prendre des mesures en conséquence pour les désengorger : retard au décollage, modification de la route à suivre, etc.

Nous modélisons (de manière très simplifiée) les plans de vol gérés par OACA sous la forme d'une base de données comportant deux tables :

– La table « **vol** » qui répertorie les plans de vol déposés par les compagnies aériennes ; elle contient les colonnes :

- **id_vol** : numéro du vol (chaîne de caractères) ;
- **depart** : code de l'aéroport de départ (chaîne de caractères) ;
- **arrivee** : code de l'aéroport d'arrivée (chaîne de caractères) ;
- **jour** : jour du vol (de type **date**, affiché au format **aaaa-mm-jj**) ;
- **heure** : heure de décollage souhaitée (de type **time**, affiché au format **hh:mi**) ;
- **niveau** : niveau de vol souhaité (entier).

id_vol	depart	arrivee	jour	heure	niveau
TU481	TUN	DJE	2020-02-29	07 :35	300
TU482	DJE	TUN	2020-02-29	10 :25	300
TU603	TUN	DJE	2020-02-29	10 :05	310
TU604	DJE	TUN	2020-02-29	13 :00	310

Figure 1 : Extrait de la table vol : vols de la compagnie Tunisair entre les aéroports Tunis Carthage (TUN) et Djerba-Zarzis (DJE)

– La table « **aéroport** » qui répertorie les aéroports tunisiens ; elle contient les colonnes :

- **IdAero** : code de l'aéroport (chaîne de caractères) ;
- **Ville** : principale ville desservie (chaîne de caractères) ;
- **Gouvernorat** : gouvernorat dans lequel se situe l'aéroport (chaîne de caractères).

<u>IdAero</u>	Ville	Gouvernorat
TUN	Tunis	Tunis
MIR	Monastir	Monastir
TBJ	Tabarka	Jendouba
DJE	Djerba	Medenine

Figure 2 Extrait de la table aeroport

Les types SQL **date** et **time** permettent de mémoriser respectivement un jour du calendrier et une heure du jour. Deux valeurs de type **date** ou de type **time** peuvent être comparées avec les opérateurs habituels (=, <, <=, etc.). La comparaison s'effectue suivant l'ordre chronologique. Ces valeurs peuvent également être comparées à une chaîne de caractères correspondant à leur représentation externe ('aaaa-mm-jj' ou 'hh:mi').

TRAVAIL DEMANDE :

- 1- **(1pt)** Donner les noms d'au moins trois exemples d'un SGBD.
- 2- **(1,5pts)** Expliquer c'est quoi un schéma relationnel dans une base de données.
- 3- **(1,5pts)** Expliquer c'est quoi une clé étrangère dans une base de données.
- 4- **(1,5pts)** En utilisant le module sqlite3 donner le script PYTHON permettant de créer la table **aeroport** dans la base "trafic.db" en exprimant toutes les contraintes d'intégrité mentionnées ci-dessus.
- 5- **(1,5pts)** Donner la/les requête(s) SQL qui permet d'ajouter dans la relation « aeroport » les deux enregistrements suivants :
 - Aéroport international d'Enfidha-Hammamet = (NBE, Enfida, Sousse)
 - Aéroport international de Sfax-Thyna = (SFA, Sfax, Sfax)
- 6- **(2pts)** Écrire une requête SQL qui fournit le nombre de vols qui doivent décoller dans la journée du 29 février 2020 avant midi.
- 7- **(2pts)** Donner l'expression en algèbre relationnelle qui fournit la liste des numéros de vols au départ d'un aéroport desservant Tunis le 29 février 2020.
- 8- **(2pts)** Traduire l'expression relationnelle de la question précédente (Question 7) en requête SQL.
- 9- **(2pts)** Donner la requête qui permet de trouver les numéros des vols et les codes des aéroports pour tous les vols de 29 février 2020 et dont l'aéroport de départ se trouve dans une ville différente de la ville principale du gouvernorat.
- 10- **(2pts)** Donner la requête qui permet de trouver tous les vols de 29 février 2020 dont le niveau du vol est inférieur à la moyenne de tous les vols de la même journée.
- 11- **(3pts)** Certains vols peuvent engendrer des conflits potentiels : c'est par exemple le cas lorsque deux avions suivent un même trajet, en sens inverse, le même jour et à un même niveau. Écrire une requête SQL qui fournit la liste des couples (Id1, Id2) des identifiants des vols dans cette situation.

***** Bon courage *****

Université de Carthage Institut Préparatoire aux Etudes d'Ingénieur de Nabeul Département de Mathématique et de l'Informatique	 المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2019/2020 Filière : SM/SP/ST Niveau d'étude : 2^{ème} année Semestre : 2 / Date : 20/02/2020 Nombre de pages : 2 Durée : 1h30
--	---	--

Devoir Surveillé de : Informatique

***** CORRIGE *****

1- (1 Point) Donner les noms d'au moins trois exemples d'un SGBD :

- Oracle, DB2, MySQL, PostgreSQL, Access, SQL Server, SQLite.

2- (1.5 Points) Expliquer c'est quoi un schéma relationnel dans une base de données :

Un schéma relationnel S est donné par une famille finie d'attributs:

$$S = (A_1; \dots; A_n) \in A^n, \text{ avec } i \neq j \Rightarrow A_i \neq A_j,$$

et « D » l'application domaine associée.

On le note aussi $S = ((A_1; D(A_1)); \dots; (A_n; D(A_n)))$

Exemple: Bibliotheque = ((titre,string),(auteur,string),(annee,int))

3- (1.5 Points) Expliquer c'est quoi une clé étrangère dans une base de données :

Dans une base de données, un lien entre deux tables A et B est représenté par l'ajout dans la table B d'une nouvelle colonne correspondant à la clé primaire de la table A.

- cette nouvelle colonne est appelée clé étrangère dans la table B.
- ou c'est une clé primaire qui migre de la table mère A vers la table fille B.
- Pour la distinguer on la précède par "#".

4- (1.5 points)

```

1. import sqlite3
2. base = sqlite3.connect('''trafic.db''')
3. cr = base.cursor()
4. cr.execute('''CREATE TABLE aeroport(
5. Id_aero TEXT PRIMARY KEY,
6. ville TEXT,
7. gouvernorat TEXT;''')
8. cr.commit()
9. cr.close()
10. base.close()

```

5- (1.5 points)

```

1. INSERT INTO aeroport VALUES('NBE', 'Enfida', 'Sousse');
2. INSERT INTO aeroport VALUES('SFA', 'Sfax', 'Sfax');

```

6- (2 points)

L'énoncé spécifie que l'on peut comparer une date et une heure avec une chaîne de caractères équivalente. De plus, l'opérateur d'agrégation permettant de compter le nombre d'enregistrements est COUNT(*). On peut ainsi écrire :

```

1. SELECT COUNT(*) FROM vol WHERE jour='2020-02-29' AND heure<'12:00';

```

7- (2 points)

```
1.  $\Pi_{id\_vol} (\sigma_{(ville='Tunis' \text{ et } jour='2020-02-29')}(vol[depart=idAero]aeroport))$ 
```

8- (2 points)

```
1. SELECT id_vol
2. FROM vol JOIN aeroport ON depart=IdAero
3. WHERE ville = 'Tunis' AND jour = '2020-02-29';
```

9- (2 points)

```
1- SELECT IdAero, id_vol FROM vol
2- JOIN aeroport ON IdAero = depart
3- WHERE Gouvernorat <> Ville AND jour = '2020-02-29';
```

10- (2 points)

```
1. SELECT * FROM vol
2. WHERE jour = '2020-02-29' AND niveau < (
3.     SELECT AVG(niveau) FROM vol
4.     WHERE jour = '2020-02-29');
```

11- (3 points)

Il est nécessaire d'utiliser deux fois la table vol et il faut les renommer. Les conditions de jointure et de sélection sont ici exceptionnellement proches, au point que plusieurs solutions sont possibles en déplaçant les conditions du ON vers le WHERE ou inversement. Par exemple, en considérant que l'on joint les tables pour les vols ayant lieu le même jour au même niveau, puis que l'on sélectionne les vols ayant les mêmes points de départ et d'arrivée, on a :

```
1. SELECT v1.id_vol AS id1, v2.id_vol AS id2
2. FROM vol AS v1 JOIN vol AS v2
3. ON v1.jour=v2.jour AND v1.niveau=v2.niveau
4. WHERE v1.depart=v2.arrivee AND v1.arrivee=v2.depart
5. AND v1.id_vol<v2.id_vol;
```

Université de Carthage Institut Préparatoire aux Etudes d'ingénieur de Nabeul Département Mathématiques et Informatique		Année universitaire : 2018/2019
		Filières : M/P/T
		Niveau d'études : 2ème année
		Semestre : 2
		Nombre de pages : 5
		Date : 23/5/2019
		Durée : 2h

Concours blanc

Exercice 1 : Programmation orientée objet

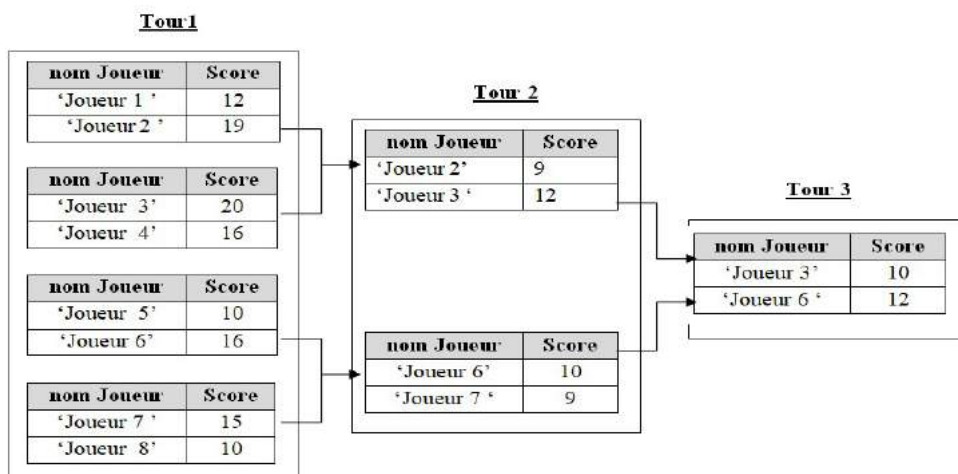
(6.5 points)

Dans cet exercice on s'intéresse à la gestion des tournois (tennis, échecs, etc.) et à l'attribution des points aux adhérents (participants).

Un tournoi est organisé en "tours". On joue les tours les uns après les autres, un tour ne pouvant être commencé avant que tous les matchs du tour précédent ne soient terminés. Chaque tour comporte un nombre de matchs égal à une puissance de 2. Les matchs d'un tour sont calculés en fonction des résultats du tour précédent : **le premier match oppose les vainqueurs des matchs 1 et 2 du tour précédent, le second les vainqueurs des matchs 3 et 4, etc.**

A chaque tour on a donc 2 fois moins de matchs qu'au tour précédent. On parle de $\frac{1}{16}$ ^{ème} lorsqu'il y a 16 matchs à jouer, puis $\frac{1}{8}$ ^{ème} pour les 8 matchs du tour suivant et ainsi de suite jusqu'aux $\frac{1}{2}$ ^{ème} finales et enfin la finale dont le vainqueur est le vainqueur du tournoi.

La figure suivante illustre le principe décrit en dessus :



On créera alors un ensemble de classes permettant d'effectuer la gestion des tournois :

Partie 1 : classe Match

Créer une classe **Match** permettant de représenter un seul match du tournoi :
Le constructeur de cette classe admet 4 attributs :

- **Numero** : un entier qui représente le numéro du match
- **Tour** : le tour du match: un entier (1 pour le premier tour , 2 pour le deuxième...)
- **LJ**= [nom_Joueur_1, nom_Joueur_2], une liste de chaines, contenant les noms des deux joueurs adversaires du match
- **LS**= [Score_joueur_1, Score_joueur_2], qui est initialement vide, une liste d'entiers, qui contient le score de chacun des joueurs à la fin du match

Cette classe admet deux méthodes :

- `__str__` permettant d'afficher un match selon le modèle suivant
'le match numéro « **num** » du tour « **tour** » oppose les joueurs « **nom_j1** » et « **nom_j2** » '
 - `Vainqueur()` permettant de retourner le joueur gagnant, une fois le match est joué
1. Créer la classe **Match**, son constructeur et ses deux méthodes décrites en dessus

Partie 2 : Une classe tournoi :

La classe Tournoi permet de modéliser les tournois. Le constructeur de cette classe initialise les attributs suivants :

- **LJ**= Liste de chaines, qui contient les noms des joueurs participant au tournoi. On supposera que la taille de cette liste est une puissance de 2, sans le vérifier.
 - **NbMatch** : de type entier, **initialement nul** et il représente le nombre total des matchs joués pendant le tournoi
 - **LM**= Liste de listes, dont chaque sous liste **LM[i]** est une liste de Match joués dans le **ième tour**. Cette liste sera **initialement Vide**
2. Définir la classe Tournoi ainsi que son constructeur
 3. Ecrire la méthode **PremierTour** permettant de créer la liste des **Matches** du premier tour et l'ajouter à la liste des Matches. Cette liste est créé par tirage au sort (aléatoirement) des joueurs
Rappel : la fonction **choice** du module **random** permet de choisir un élément aléatoirement depuis une liste
 4. Ecrire une méthode **Affiche** qui affiche tous les matchs d'un tour « **t** » passé comme paramètre
 5. Ecrire une méthode **TourSuivant**, permettant de créer la liste des **Matches** du tour suivant
NB : les matchs ayant été joués les scores sont alors mis à jour
 6. Ecrire une méthode **Champion** qui, une fois tous les matchs sont joués retourne le nom du joueur gagnant
 7. Ecrire la méthode **Statistique_Joueur** qui une fois le tournoi est terminé, prend comme paramètre le nom d'un joueur **nomJ** et retourne une liste **LJ** contenant les numéros des Matches où il a participé

Exercice 2 : Base de Données

(5.5 points)

Soit le schéma relationnel de la base de données « **ParcCamion.db** » qui représente la gestion d'un parc de camions permettant de transporter des marchandises vers différentes destinations (villes)

Type(IdType, Marque, modele, volumeMax)

La table Type contient les différents types des camions du parc

- **IdType** : identifiant du type (type chaine), clé primaire
- **Marque** : marque d'un camion (type chaine)
- **Modele** : le modèle d'un type, comme par exemple 'ISUZU', 'Ford'... (une chaine)
- **volumeMax** : volume max qui peut être transporter par ce type (un entier)

Camion(immatriculation, DateMiseService, etat, IdType#)

La table camion représente tous les camions du parcs

- **Immatriculation** : l'immatriculation d'un camion (chaine de caractères), clé primaire
- **dateMiseService** : la date de mise en service du camion (une date)
- **etat** : une chaine ∈ ('nouveau', 'bon etat', 'etat moyen', 'detruit') qui représente l'etat d'un camion
- **IdType** : l'identifiant du type auquel appartient camion, clé étrangère

Marchandise(code, designation, volume , poids)

La table marchandise contient les marchandises transportées par les camions du parc

- Code : le code de la marchandise (un entier), clé primaire
- Designation : la désignation de la marchandise (chaîne de caractère)
- volume : le volume de la marchandise (un entier)
- poids : le poids de la marchandise (un réel)

Livraison (code#,immatriculation#,ville, Distance, dateLiv)

La table livraison représente les différentes livraisons des marchandises effectuées par les camions

- Code : le code de la marchandise livrée, clé étrangère
- Immatriculation : l'immatriculation du camion qui a effectué la livraison, clé étrangère
- Ville : la destination de la livraison (chaîne de caractère)
- Distance : la distance entre le parc et la ville destination (un réel)
- dateLiv : la date de la livraison (une date)

1. donner les instructions nécessaires pour se connecter à la base « **ParcCamion.db** » et créer le curseur.
2. Ecrire une fonction **SupprimeDetruit** qui supprime de la base tous les camions dont l'état est « **détruit** »
3. Donner les requêtes SQL permettant de :
 - 3.1.Afficher les immatriculations, le modèle et le volume maximal des camions mis en service en 2018
 - 3.2.Afficher, sans répétition, les volumes qui peuvent être transportés par les camions
 - 3.3.Compter le nombre des livraisons effectué par chaque camion
 - 3.4.Afficher les camions qui ont parcouru une distance totale supérieure à 100 000 Km
 - 3.5.Afficher la désignation, le volume et le poids des marchandises ainsi que la date de livraison de toutes les marchandises livrées à « Tunis ».

Exercice 3 :numpy

(4 points)

Dans cet exercice, on se propose d'étudier une matrice spéciale appelée « **matrice unimodulaire** » de type **matrice combinatoire**

Soit A une matrice de nombre de lignes p et de nombre de colonnes q tel que $p \leq q$ dont tous les coefficients sont des 0, des 1 ou des -1. On dit que A est **unimodulaire** si et seulement si toutes les sous-matrices carrées d'ordre (p, p) ont un déterminant égal à 0, 1 ou -1.

On dit que A est **totalelement unimodulaire** si et seulement si le déterminant de toutes les sous-matrices carrées de A vaut 0, 1 ou -1.

On obtient des matrices totalement unimodulaires par différentes méthodes:

- 1) Si chaque ligne de A contient un seul coefficient égal à un 1 ou à -1 (les autres coefficients étant nuls), alors A est totalement unimodulaire

- 2) Si A est une matrice unimodulaire alors la matrice, obtenue en concaténant A à I_p ,

$$(A, I_p) = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0,q-1} & 1 & 0 & \dots & 0 \\ a_{10} & a_{11} & \dots & a_{1,q-1} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{p-1,0} & a_{p-1,1} & \dots & a_{p-1,q-1} & 0 & 0 & \dots & 1 \end{pmatrix} \text{ est totalement unimodulaire, où } I_p \text{ est la matrice identité d'ordre } p$$

- 3) Toute matrice **équilibrée** est une matrice totalement unimodulaire :

Soit une matrice A d'ordre (p, q) , dont tous les coefficients sont des 0, des 1 ou des -1. On note C_j la colonne d'indice j de A pour $0 \leq j < q$.

On dit qu'une matrice A est équilibrée si toute différence $d = \sum_{j=0}^k C_j - \sum_{j=k+1}^{q-1} C_j$ est égale à 0, 1 ou -1 $\forall k \in \{0..q-2\}$

Exemple : pour $q=4$

- $K=0, d = \sum_{j=0}^0 C_j - \sum_{j=1}^3 C_j$
- $K=1, d = \sum_{j=0}^1 C_j - \sum_{j=2}^3 C_j$
- $K=2, d = \sum_{j=0}^2 C_j - \sum_{j=3}^3 C_j$

Travail Demandé

NB : pour toutes les questions, on suppose disposer d'une matrice $A(p,q)$ à coefficients dans $\{0,1,-1\}$

1. Ecrire une fonction **Unimodulaire(A,p,q)** qui teste si A unimodulaire ou non
2. Ecrire une fonction **Total_Unimod1(A,p,q)**, qui vérifie si chaque ligne de A contient exactement un seul +1 ou un seul -1 (les autres coefficients étant nuls)
3. Ecrire une fonction **Total_Unimod2(A,p,q)**, qui prend comme paramètre une matrice A , supposée unimodulaire et retourne la matrice (A, I_p) décrite par 2) page 3/5
4. Ecrire une fonction **Différence(A,p,q,K)**, qui calcule la différence $d = \sum_{j=0}^k C_j - \sum_{j=k+1}^{q-1} C_j$
5. Ecrire une fonction **Total_Unimod3(A,p,q)**, qui vérifie si A est équilibrée selon le principe décrit en dessus.

Exercice 4 : Image & Cryptographie

(5.5 points)

La **stéganographie** est une branche de la cryptographie qui consiste à dissimuler (cacher) un message dans un autre.

Le but de cet exercice est de cacher un message quelconque (un texte clair) au sein d'une image en niveaux de gris. Où chaque pixel est codé sur 1 octet=8 bits.

Exemple

Un pixel dont la valeur est 63 est représenté par : [0, 0, 1, 1, 1, 1, 1, 1]

0	0	1	1	1	1	1	1
plus fort				plus faible			

Évidemment, il ne s'agit pas d'écrire le texte en plein milieu du dessin, ni d'ajouter du texte en commentaire dans l'image. L'idée est plutôt de **modifier les deux bits de poids le plus faibles des pixels par un bout de lettre...**

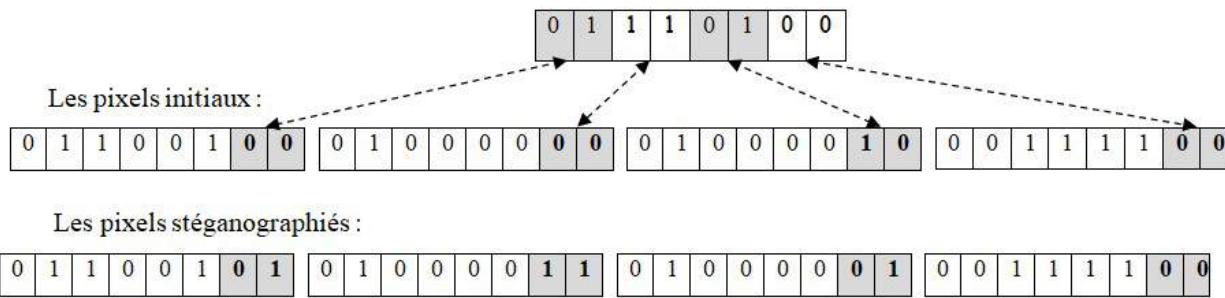
Principe :

- Extraire les pixels de l'image dans une liste d'entiers **Lpixel**
 - Convertir les pixels de la liste **Lpixel** de la base 10 en base 2
 - Modifier les valeurs des pixels de l'image un par un comme suit:
 - Chaque caractère du texte à cacher sera représenté par l'écriture binaire de son code ascii
 - chaque caractère sera codé sur quatre pixels successifs : **2 bits par pixel**
 - les bits qui seront modifiés sont les bits de poids les plus faibles (à droite), de telle sorte que le changement aura un effet négligeable sur l'image
 - on ajoute un caractère spécial (qui n'existe pas dans le texte) permettant de détecter la fin du texte à cacher
- Exemple :** Texte='Programmation Python', caractère='*', on dissimule alors les caractères de texte puis le caractère '*'
- reconstruire une nouvelle image à partir de la nouvelle liste des pixels modifiée.

Exemple :

Soit les quatre pixels : 100, 64, 66, 60 et le caractère « t » dont le code ascii est 116

« t » est représenté par : $(116)_{10} = (01110100)_2$

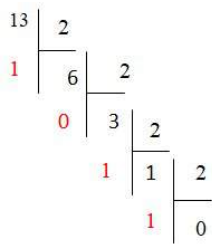


Travail demandé :

NB : On suppose avoir définie les deux fonctions suivantes :

- **Décomposer** : une fonction qui convertit un entier **nb** en base 10 en son équivalent binaire.
NB : **nb** est compris entre 0 et 255 , le résultat est une liste de taille 8 composée de 0 et de 1.

Exemple pour **nb**=13, on aura



Bit=[0,0,0,0,1,0,1,1]

- **Décimale** : une fonction qui convertit une liste composée de 0 et de 1 en son en base 10
1. Ecrire une fonction **Extraire**, qui extrait les pixels de l'image stockée dans le fichier portant le nom« **nom_im** » en une liste contenant tous les pixels .
 2. Ecrire une fonction **EcritureBinaire** qui prend comme paramètre une liste d'entiers L et retourne une deuxième liste LB tel que **chaque élément de LB est une liste** qui représente la représentation binaire d'un élément de L
Exemple : pour L=[60,113] on retourne[[0, 0, 1, 1, 1, 1, 0, 0], [0, 1, 1, 1, 0, 0, 0, 1]]
 3. Ecrire une fonction **DissimulerCar** qui prend comme paramètre 2 listes : **Bit**, contenant la représentation binaire d'un caractère est L=[**px1,px2,px3,px4**]contenant les représentations binaire de **quatre pixels** et retourne une liste L1 contenant les pixels modifiés selon le principe décrit en dessus
Exemple :
Bit=[0, 1, 1, 1, 0, 0, 0, 1]
L=[[0,1,1,0,0,1,0,0],[0,1,0,0,0,0,0,0],[0,1,0,0,0,0,1,0],[0,0,1,1,1,1,0,0]]
On obtient
L1=[[0,1,1,0,0,1,**0,1**],[0,1,0,0,0,0,**1,1**],[0,1,0,0,0,0,**1,0**],[0,0,1,1,1,1,**0,1**]]
 4. Ecrire une fonction **DissimulerTexte** qui prend comme paramètre une liste de pixel **Lpixel** ,un texte clair **Texte** (une chaine de caractères alphabétique)et un caractère spécial **car** , et retourne une liste **Lpixel2**,contenant les écritures binaires des pixels selon le principe décrit en dessus
NB :on supposera que la taille de **Lpixel** est suffisante pour coder le texte et le caractère spécial de la fin
 5. Ecrire une fonction **ReconstruireImage(Lpixel2,larg, haut)** qui crée et affiche une nouvelle image à partir de la liste **Lpixel2**, obtenue par la fonction en 4)

Correction Concours Blanc 2018-2019

Exercice 1

1.

```
class Match:
```

```
    def __init__(self,num,tour,LJ):
```

```
        self.numero=num
```

```
        self.tour=tour
```

```
        self.LJ=LJ
```

```
        self.LS=[]
```

```
    def __str__(self):
```

```
        F='le match numéro {} du tour {} oppose les joueur {} et {}'
```

```
        return(F.format( self.numero,self.tour,self.LJ[0],self.LJ[1]))
```

```
    def Vainqueur(self):
```

```
        if self.LS[0]>self.LS[1]: return(self.LJ[0])
```

```
        else : return(self.LJ[1])
```

2.

```
from random import choice
```

```
from copy import copy
```

```
class Tournoi:
```

```
    def __init__(self,Joueur):
```

```
        self.LJoueur=Joueur
```

```
        self.NbMatch=0
```

```
        self.LM=[]
```

3.

```
    def PremierTour(self):
```

```
        L=[]
```

```
        L1=copy(self.LJoueur)
```

```
        #le nombre des itérations est nb_joueur//2
```

```
        for i in range(len(self.LJoueur)//2):
```

```
            # choisir un nom et le supprimer pour ne pas le prendre une autre fois
```

```
            j1=choice(L1)
```

```
            L1.remove(j1)
```

```
            j2=choice(L1)
```

```
            L1.remove(j2)
```

```
            # incrémenter le nombre des matchs : c'est le numéro de match
```

```
            self.NbMatch+=1
```

```
            # créer un objet Match du tour 1
```

```
            M=Match(self.NbMatch,1,[j1,j2])
```

```
            L.append(M)
```

```
        # ajouter la liste L à l'attribut self.LM : contenant les matchs de chaque tour
```

```
        self.LM.append(L)
```

4.

```
def Affiche(self,t):  
    # la liste des match du tour 't' se trouve dans l'indice 't-1'  
    for m in self.LM[t-1]:  
        print(m) # appel à la méthode spéciale __str__ de la classe Match
```

5.

```
def TourSuivant(self):  
    # Extraire la liste des Match du tour précédent  
    L1=self.LM[-1]  
    tour=len(self.LM)+1  
    L=[]  
    for i in range(0,len(L1),2):  
        # calculer les vainqueurs de chaque 2 matchs successifs  
        v1=L1[i].Vainqueur()  
        v2=L1[i+1].Vainqueur()  
        # incrémenter le nombre des matchs  
        self.NbMatch+=1  
        # créer un objet match et l'ajouter  
        M=Match(self.NbMatch,tour,[v1,v2])  
        L.append(M)  
    self.LM.append(L)
```

6.

```
def champion(self):  
    # le champion est le vainqueur du MATCH du dernier tour  
    # Attention : le dernier tour est une liste contenant un seul Match  
    return (self.LM[-1][0].Vainqueur())
```

7.

```
def StatistiqueJoueur(self,J):  
    L=[]  
    # parcourir tous les tours  
    for e in self.LM:  
        # parcourir tous les Matches de ce tour  
        for m in e:  
            if J in m.LJ: L.append(m.numero)  
    return(L)
```

Exercice 2

1.

```
import sqlite3  
conn=sqlite3.connect('ParcCamion.db')  
cur=conn.cursor()
```

2.

```
def SprimeDetruit():  
    req="DELETE FROM Camion WHERE etat='détruit' "
```

```
cur.execute(req)
```

```
conn.commit()
```

3.

```
3.1.req="" SELECT immatriculation,Modele, VolumeMax
FROM Camion as C, Type as T
WHERE C.IdType=T.IdType
AND dateMiseService BETWEEN '01/01/2018' AND '31/12/2018' ""
```

```
3.2.req='SELECT DISTINCT volumeMax FROM Type'
```

```
3.3.req=""SELECT COUNT(*), immatriculation
FROM Livraison
GROUP BY immatriculation ""
```

```
3.4.req=""SELECT SUM(Distance), immatriculation
FROM Livraison
GROUP BY immatriculation
HAVING SUM(Distance)>100000""
```

```
3.5.req=""SELECT Designation, volume,poids, dateLiv
FROM Marchandise as M, Livraison as L
WHERE M.code=L.code
AND ville='Tunis' ""
```

Exercise 3

```
from numpy import *
```

```
from numpy.linalg import det
```

1.

Exemple pour $A = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$

Les déterminants des matrices $SM1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$, $SM2 = \begin{pmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$ doivent $\in \{0,1,-1\}$

```
def Unimodulaire(A,p,q):
```

```
    for i in range(q-p+1):
```

```
        # extraire une sous matrice puis calculer son déterminant
```

```
        SM=A[:, i:i+p]
```

```
        if det(SM) not in (0,1,-1): return(False)
```

```
    return(True)
```

2.

```
def TotalUnimod1(A,p,q):
```

```
    for i in range(p):
```

à chaque itération on doit extraire la ième ligne
#le nombre des 0 doit être égal au nombre des élément-1
(l'autre élément peu être soit 1 ou -1)

```
T=A[i ,:]
T1=where(T==0)
if T1.size!=T.size-1: return(False)
return(True)
```

3.

```
def TotalUnimod2(A,p,q):
    # créer la matrice identité puis la concaténer à la matrice A avec la fonction append
    Ip=identity(p)
    return(append(A,Ip,axis=1))
```

4.

```
def Différence(A,p,q,K):
    SM1=A[:, :k+1]
    SM2=A[:, k+1 :]
    Return(sum(SM1)-sum(SM2))
```

5.

```
def Total_Unimod3(A,p,q):
    for K in range(0,q-1):
        if Différence(A,p,q,K) not in (1,-1,0) :      return(False)
    return(True)
```

Exercice 4

```
from PIL.Image import *
```

1.

```
def Extraire(nom):
    Im=open(nom)
    Lp=list(Im.getdata())
    return(Lp)
```

2.

```
def EcritureBinaire(L):
    return([Decomposer(e) for e in L])
```

3.

Exemple : Bit=[0, 1, 1, 1, 0, 0, 0, 1]
 L=[[0,1,1,0,0,1,0,0],[0,1,0,0,0,0,0,0],[0,1,0,0,0,0,1,0],[0,0,1,1,1,1,0,0]]

On a 4 itérations pour cacher Bit dans L

- **i=0**
 L[0]= **[0,1,1,0,0,1,0,0]**, on ne prend que les 6 premier éléments → **[0,1,1,0,0,1]**
 on extrait les deux premiers éléments de Bits → **Bit[0:2]**
on crée la nouvelle sous liste par la concaténation [0,1,1,0,0,1]+ Bit[0:2]
- **i=1**
 L[1]= **[0,1,0,0,0,0,0,0]** on ne prend que les 6 premier éléments → **[0,1,0,0,0,0]**
 on extrait les deux éléments suivants de Bits → **Bit[2:4]**

on crée la nouvelle sous liste par la concaténation $[0,1,0,0,0,0]+Bit[2:4]$

$\rightarrow Bit[i*2:i*2+2]$

- $i=2$

$L[2]=[0,1,0,0,0,0,1,0]$ on ne prend que les 6 premier éléments $\rightarrow [0,1,0,0,0,0]$

on extrait les deux éléments suivants de Bits $\rightarrow Bit[4:6]$

on crée la nouvelle sous liste par la concaténation $[0,1,0,0,0,0]+Bit[4:6]$

$\rightarrow Bit[i*2:i*2+2]$

- $i=3$

$L[3]=[0,0,1,1,1,1,0,0]$ on ne prend que les 6 premier éléments $\rightarrow [0,0,1,1,1,1]$

on extrait les deux éléments suivants de Bits $\rightarrow Bit[6:8]$

on crée la nouvelle sous liste par la concaténation $[0,0,1,1,1,1]+Bit[6:8]$

$\rightarrow Bit[i*2:i*2+2]$

```
def DissimulerCar(Bit,L):
    L1=[]
    for i in range (4):
        ancien=L[i]
        nouv=ancien[:6]+Bit[i*2:i*2+2]
        L1.append(nouv)
    return(L1)
```

Remarque :

On peut prendre à chaque itération, les deux premiers éléments de Bits puis on les supprime

```
from copy import copy
def DissimulerCar(Bit,L):
    L1,Bit1=[], copy(Bit)
    for i in range (4):
        ancien=L[i]
        nouv=ancien[:6]+Bit1[:2]
        Bit1=Bit1[2:]
        L1.append(nouv)
    return(L1)
```

4.

Exemple :

Soit LPbin la liste qui représente l'écriture binaire des pixels

- $i=0 \rightarrow LPbin[0:4]$
- $i=0 \rightarrow LPbin[4:8] \rightarrow LPbin[i*4:i*4+4]$
- $i=0 \rightarrow LPbin[8:12]$
- ...

```

def Dissimuler(texte,Lpixel,car):
    # transformer le texte en une écriture binaire : liste de listes
    L=[ord(e) for e in texte]
    LTbin=EcritureBinaire(L)
    # transformer la liste des pixels en une écriture binaire : liste de listes
    LPbin=EcritureBinaire(Lpixel)
    n=len(LTbin)
    L1=[]
    # Parcourir le texte , qui est en écriture binaire
    for i in range(n):
        #à chaque itération on extrait les sous listes de quatre pixels
        SL=LPbin[i*4:i*4+4]
        L1+=DissimulerCar(LTbin[i],SL)
    #ajouter le caractère spécial
    L1+=DissimulerCar(Decomposer(ord('*')),LPbin[n*4:n*4+4])
    return(L1)

```

Remarque :

On peut prendre à chaque itération, les quatre premières sous listes de **LPbin** puis on les supprime

```

def Dissimuler(texte,Lpixel,car):
    # transformer le texte en une écriture binaire : liste de listes
    L=[ord(e) for e in texte]
    LTbin=EcritureBinaire(L)
    # transformer la liste des pixels en une écriture binaire : liste de listes
    LPbin=EcritureBinaire(Lpixel)
    LPbin1=copy(LPbin)
    n=len(LTbin)
    L1=[]
    for i in range(n):
        #à chaque itération on extrait les quatre sous listes puis on les supprime
        SL=LPbin1[:4]
        LPbin1= LPbin1[ 4 :]
        L1+=DissimulerCar(LTbin[i],SL)
    #ajouter le caractère spécial
    L1+=DissimulerCar(Decomposer(ord('*')),LPbin[n*4:n*4+4])
    return(L1)

```

5.

```

def ReconstruireImage(npmF,Lpixel2,larg,haut):
    L=[Decimale(e) for e in Lpixel2]
    Im=new('L',(larg,haut))
    Im2.putdata(L)
    Im2.show()

```

Université de Carthage Institut Préparatoire aux Etudes d'ingénieur de Nabeul Département :mathématique		Année universitaire : 2019/2020
		Filière : SM/SP/ST
		Niveau d'étude : 2 ^{ème} année
		Semestre :1
		Nombre de pages :2
		Date : 14/11 / 2019 Durée :1h30

Devoir Surveillé de : Informatique

Exercice 1 : Programmation Orientée Objet (12 pts)

Le jeu de Marienbad

Le jeu de Marienbad appelé aussi le jeu des allumettes, c'est un jeu qui nécessite deux joueurs et 21 allumettes.

Au début, Les 21 allumettes sont réparties en 6 tas sur la table de jeu de la manière suivante :Une allumette dans le premier tas, deux allumettes dans le deuxième tas, etc.. c'est à dire i allumettes dans le i^{ème} tas.

Exemple :

```

Tas 1 :      |
Tas 2 :      | |
Tas 3 :      | | |
Tas 4 :      | | | |
Tas 5 :      | | | | |
Tas 6 :      | | | | | |

```

Principe de jeu :

Chaque joueur peut prendre un nombre n d'allumettes souhaitées dans un seul tas .

Chacun à son tour jusqu'à vider la table de jeu.

Le joueur qui prend la dernière allumette perd la partie.

Travail demandé

On souhaite écrire un programme qui fait jouer deux joueurs humains représentés par leurs noms.au cours du jeu, l'affichage se présentera sous la forme suivante :

Tas : (1, 2, 3, 4, 5, 6)

Allumettes : [0, 2, 0, 4, 5, 3]

Prochain joueur : Sami

1) Définir une classe nommée **Marienbad_partie** qui comprend :

- Les attributs suivants :
 - ✓ **Tas** : attribut privé qui est un tuple initialisé aux numéros des tas présentés dans ce jeu.
 - ✓ **AL** : une liste d'entier qui représente l'état de la table de jeu après chaque tour joué, cette liste est initialisée à l'état initial des tas.
 - ✓ **Joueurs** : un tuple de chaîne de caractères contenant les noms des 2 joueurs.
 - ✓ **Tour** : un entier (1 ou -1) qui permet d'alterner le joueur1 et le joueur2 à chaque tour, cet attribut est initialisé à 1.
- Les méthodes suivantes :
 - ✓ **Un constructeur** avec des noms de joueurs par défaut «joueur1» et «joueur2».
 - ✓ **affich_etat()** qui affiche les informations d'état de jeu à chaque tour.
 - ✓ **verifie()** qui teste s'il est possible d'enlever un nombre **n** d'allumettes d'un tas numéro **t** présent.
 - ✓ **enlever()** qui met à jour l'état de jeu en enlevant **n** allumettes dans le tas **t** sans vérification.
 - ✓ **termine()** qui renvoie un booléen indiquant si le jeu est terminé ou non.

2) Définir une classe nommée **Marienbad** qui représente toutes les parties Marienbad jouées le long de la journée dans la salle de jeux.

Cette classe est définie par :

- Un seul attribut **DM** de type dictionnaire initialement vide qui a comme :
 - Clé : tuple contenant les noms des 2 joueurs de la partie Marienbad.
 - Valeur : chaîne de caractère présentant le nom du gagnant de cette partie.
- Une méthode **jouer()** qui permet de faire tourner une partie Marienbad avec deux noms de joueurs saisis au clavier et stocker le résultat dans **DM**.
- Une méthode **gagnant()** qui renvoie une liste contenant les noms des gagnants le long d'une journée.

3) Ecrire un programme principal qui permet de lancer 10 parties d'un jeu Marienbad et d'afficher la liste des noms des gagnants.

Le programme doit afficher aussi les noms des gagnants qui auront des prix. Un gagnant aura un prix s'il gagne plus qu'une partie du jeu Marienbad.

Le Jeu Mastermind

Le jeu Mastermind ou appelé aussi Taureaux et vaches nécessite deux joueurs et six boules de couleurs différentes.

Chaque joueur possède un support à 4 cases à remplir avec 4 boules de couleurs différents, chaque remplissage est un code secret pour son joueur.

Chaque joueur tente, à son tour, de deviner le code secret de son adversaire.

L'adversaire doit marquer le coup : « **point noir** » (appelé aussi Taureau) pour une boule de couleur située à la bonne position et « **point blanc** » (appelé aussi Vache) pour une boule de couleur existante mais située à une mauvaise position dans le code secret de l'adversaire.

Le premier joueur qui détermine le code secret de l'autre est le gagnant de la partie.

Mise en Œuvre

On souhaite écrire un programme python permettant de stimuler le jeu Mastermind entre l'utilisateur du jeu comme premier joueur et la machine comme un deuxième joueur (c'est-à-dire l'adversaire).

Pour ce faire, on suit le principe des données suivant :

- Le programme choisi au hasard un code secret (qui sont les 4 couleurs de boules) parmi les 6 couleurs de la liste **Couleurs = ['rouge','vert','bleu','jaune','rose','orange']**.
Le code secret est alors une liste appelée **LS à 4 couleurs** différentes.
- L'utilisateur doit saisir son essai dans une liste de 4 couleurs différentes appelée **LE**, un essai saisi ne doit pas être testé avant.
- Après chaque essai, le programme doit afficher sur l'écran le nombre de point noirs **nb_noir** et le nombre de points blancs **nb_blanc** résultant de l'essai.
- Chaque essai doit être stocké dans une liste appelée **DEVINE** .
Devine est une liste de liste, chaque sous liste est composée de deux éléments : le premier élément et une liste d'essai et le deuxième élément est un tuple contenant le nombre de points noirs et le nombre de points blancs résultant : **(nb_noir, nb_blanc)**.

Exemple :

LS = ['jaune','vert','rose','bleu']

✓ Pour un 1^{ier} essai : **LE = ['jaune','bleu','rose','rouge']**

→ nb_noir=2 et nb_blanc=1

→DEVINE=[['jaune','bleu','rose','rouge'],(2,1)]

✓ Pour un 2^{ème} essai : **LE** = ['jaune','vert', 'rose','orange']

➔**DEVINE**=[[['jaune','bleu', 'rose','rouge'],(2,1)] ,
[['jaune','vert', 'rose','orange'],(3,0)]]

✓ Etc...

- Le programme doit afficher « c'est gagné » si l'utilisateur détermine le code secret et afficher le message « c'est perdu » dans le cas où l'utilisateur ne trouve pas le code secrets au bout de 12 tentatives.

Travail demandé

- 1) Ecrire une fonction python nommée **Liste_secret** qui permet de créer une liste **LS** de couleurs choisit au hasard par le programme à partir d'une liste **COULEURS**.
Nb : on peut utiliser la commande `choice(L)` du module `random` qui permet de choisir un élément aléatoire de la liste `L`.
- 2) Ecrire une fonction appelée **Saisie_essai** qui permet de lire et remplir une liste **LE** de couleurs choisies par l'utilisateur avec tous les contrôles nécessaires.
- 3) Ecrire une fonction python nommée **Test_essai** qui permet d'afficher le résultat du test d'une liste **LE** donnée comme essai et effectuer les mises à jour nécessaires.
- 4) Ecrire une fonction python appelée **Mastermind** qui à partir d'un code secret **LS** et une liste de couleurs **Couleurs** fait tourner le jeu Mastermind et afficher si l'utilisateur a gagné ou a perdu la partie.
- 5) Supposer que le module `pile.py` est déjà existant, ce module contient toutes les fonctions nécessaires pour manipuler une pile comme :
 - `Créer_pile()` : qui permet d'initialiser une pile vide
 - `Vide(P)` : qui teste une pile vide ou non.
 - `Empieler(p,x)` : qui permet d'ajouter un élément `x` à la pile `p`
 - `Dépiler(p)` : qui permet de faire sortir un élément de la pile `p`

Refaire une partie de la question 3) en écrivant une fonction python nommée **test_essai_pile** qui permet de compter et d'afficher le nombre de point noirs **nb_noir** à partir d'une liste de code **LS** et une liste d'essai **LE**.

NB : Les listes de départs doivent être stockées dans deux piles.

Université de Carthage Institut Préparatoire aux Etudes d'ingénieur de Nabeul Département :mathématique	 المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2019/2020
		Filière : SM/SP/ST
		Niveau d'étude : 2 ^{ème} année
		Semestre :1
		Nombre de pages :2
		Date : 14/11 / 2019
		Durée :1h30

Correction Devoir Surveillé de : Informatique

Exercice 1 : Programmation Orientée Objet (12 pts)

Le jeu de Marienbad

1)	<pre> class Marienbad_partie: # tas et AL ont une valeur initiale fixe donc on n'a pas # besoins de variables pour les initialiser # joueur est un tuple qui a une valeur par défaut→ il est # initialiser dans l'entete du constructeur def __init__(self,joueurs=('joueur1','joueur2')): self.__tas=(1,2,3,4,5,6) self.AL=[1,2,3,4,5,6] self.joueurs=joueurs self.tour=1 def affich_etat(self): print('TAS :',self.__tas) print('Allumettes:',self.AL) # le nom du joueur qui jouera le tour suivant d épend de # l'attribut »tour » if self.tour==1: print('prochain joueur:', self.joueurs[0]) else: print('prochain joueur:', self.joueurs[1]) def verifie(self,n,t): # les numéros des tas sont de 1 à 6 if t in self.__tas: # les allumettes du tas numéro « t » se trouvent dans # la case d'indice t-1 de l'attribut « LA » ->le nombre # des allumettes à tirer doit <= nombres des allumettes # de self.AL[t-1] if self.AL[t-1]>=n: return True return False def enlever(self,n,t): </pre>
----	---

	<pre> self.AL[t-1]-=n self.tour=-self.tour </pre>
	<pre> def termine(self): # la partie est termin�� quand toutes les allumettes sont # tir��es�� tous les ��l��ments de « AL » sont nuls if self.AL==[0]*6: return True return False </pre>
2)	<pre> class Marienbad: # on a un seul attribut qui est initialement vide donc le # constructeur ne prend pas de param��tre def __init__(self): self.DM=dict() def jouer(self): # introduire les noms des deux joueurs nom1=input('donner le nom du joueur1') nom2=input('donner le nom du joueur2') # d��marrer le jeux�� cr��er un objet Marienbad_partie PM=Marienbad_partie((nom1,nom2)) #tant que la partie n'est pas termin��e while PM.termine()==False: # saisir le num��ro de tas duquel on va tirer des # allumettes ainsi que le nombre �� retir�� while 1: try: t=int(input('numero de tas')) n=int(input('donner le nombre ')) # appeler la m��thode verifie pour que t et n # ont des valeurs valides if PM.verifie(n,t): break except: continue # enlever les allumettes PM.enlever(n,t) # changer le joueur if PM.tour==1: self.DM[(nom1,nom2)]=nom1 else: self.DM[(nom1,nom2)]=nom2 </pre>
	<pre> def gagnant(self): return list(self.DM.values()) </pre>
3)	<pre> M=Marienbad() for i in range(10): M.jouer() </pre>

	<pre>L=M.gagnant() print('la liste des gagnants est:',L)</pre>
	<pre>for e in L: if L.count(e)>=2: print(e)</pre>

Exercice 2 : programmation modulaire

(8 pts)

Le Jeu Mastermind

1)	<pre>def liste_secret(Couleurs): from random import choice lS=[] for i in range(4): while 1: c=choice(couleurs) # les couleurs choisies doivent être distinct if c not in ls: lS.append(c) break</pre>
2)	<pre>def saisie_essai(Couleurs,DEVINE): LE=[] while 1: # créer une liste de 4 couleurs différentes for i in range(4): while 1: c=input('donner une couleur') if c in Couleurs and c not in LE: LE.append(c) break # Ltest contient tous les essais précédents Ltest=[l[0] for l in DEVINE] # on ne doit pas introduire le même essai plus qu'une fois cette liste ne doit pas figurée dans la liste devine if LE not in Ltest: return LE</pre>
3)	<pre>def test_essai(LE,LS,DEVINE): nb_noir=0 nb_blanc=0 for i in range(len(LE)): if LE[i]==LS[i]: nb_noir+=1 elif LE[i] in LS: nb_blanc+=1 DEVINE.append([LE,(nb_noir,nb_blanc)])</pre>

	<code>print(nb_noir,nb_blanc)</code>
4)	<pre> def Mastermind(Couleurs): # la liste des essais est initialement vide DEVINE=[] # créer la liste secrète à deviner LS=liste_secret(Couleurs) i=0 # le nombre des essais ne doit pas dépasser 12 while i<=12: # saisir l'essai LE=saisie_essai(Couleurs,DEVINE) #comparer la liste LE et la liste secrète LS test_essai(LE,LS,DEVINE) t=DEVINE[-1][1] # t est le tuple(nbNoir,nbBlanc) i+=1 if t[0]== len(LS): # c'est-à-dire nbNoir=4→toutes les couleurs entre LE et LS sont identiques return True return False </pre>
5)	<pre> def test_essai_pile(LS,LE): from pile.py import * PS=creer_pile() # où on empile les éléments de LS PE=creer_pile() # où on empile les éléments de LE for i in range(4): empiler(PS,LS[i]) empiler(PE,LE[i]) nb_noir=0 while not vide(PE): if depiler(PE)==depiler(PS): nb_noir+=1 return nb_noir </pre>

Université de Carthage Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul Département Mathématique- Informatique	 المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2019/2020
		Filière : SM/SP/ST
		Niveau d'étude : 2 ^{ème} année
		Semestre : 1
		Nombre de pages : 4
		Date : 14/12/2019 Durée : 2h

Examen de : Informatique

Problème 1

(11 points)

Partie 1

L'adresse IP est un numéro servant à identifier d'une façon unique chaque ordinateur connecté à internet, elle est formée de 4 entiers positifs compris entre 0 et 255. Elle a la forme suivante :

Nombre0. Nombre1. Nombre2. Nombre3 avec $0 \leq \text{Nombre} \leq 255$

Les classes d'une adresse IP

Une adresse IP est composée de 2 parties :

- **L'identifiant du réseau** : elle correspond à l'adresse réseau dans l'Internet : elle est constituée **du ou des nombres gauches** de l'adresse IP. Il dépend de la classe de l'adresse IP . on trouve seulement trois classes : A, B ou C
- **L'identifiant machine** : elle désigne l'adresse de la machine dans le réseau : elle est constituée **du ou des nombres droits** de l'adresse IP

Le tableau ci dessous résume les caractéristiques des 3 classes A,B et C des adresses IP

classe	Identifiant réseau	Nombre0
A	Nombre0	1 à 126
B	Nombre0. Nombre1	128 à 191
C	Nombre0. Nombre1. Nombre2	192 à 223

- **Exemple** : une adresse IP de classe A, car $NB_0 = 110 \in [1..126]$, possède les caractéristiques suivantes

AdresseIP = 110. 20.46.145

Identifiant réseau identifiant machine

Travail demandé

On se propose de créer une classe **adresseIP** permettant de représenter les adresses IP selon le principe ci dessus.

Cette classe admet un attribut de classe : un dictionnaire **Dclasse** dont :

- **La clé** : une classe réseau
- **La valeur** : une liste qui représente l'intervalle des valeurs possibles de **NB₀** , pour cette classe

Exemple : **Dclasse['A'] = [1,126]**

1. Définir la classe **adresseIP** ainsi que son constructeur qui initialise quatre attributs : **NB0**, **NB1**, **NB2** et **NB3**, de type entier , représentant les différents nombres d'une adresse IP

2. Définir une méthode **Classe** permettant de retourner la classe de l'adresse IP
NB : utiliser obligatoirement l'attribut de classe **Dclasse**
3. Définir une méthode **AdrReseau** permettant de retourner une **adresseIP** représentant l'adresseIP du réseau **sachant que l'adresse réseau est obtenue en plaçant tous les nombres de l'identifiant machine à 0**
Exemple pour l'adresse IP « **110. 20.46.145** » l'adresse du réseau associé sera « **110. 0.0.0**»
4. Définir une méthode **__eq__** permettant de tester si deux **adresseIP** sont identiques
5. Définir une méthode **MemeReseau** permettant de vérifier si deux **adressesIP** appartiennent au même réseau
6. Définir une méthode **__str__** permettant d'afficher une **adresseIP** sous la forme :
« **nb0.nb1.nb2.nb3** »

Partie 2

On se propose de construire une classe **Message** qui permettra de simuler les envois des messages par Internet. Pour cela on considère une version très simplifiée dans laquelle un message contient les attributs suivants:

- **Emetteur** : l'**adresseIP** de l'émetteur
- **Date** : une chaîne de la forme « j/m/année-heure:min:sec »
- **Contenu** : le contenu du message (une chaîne de caractères)
- **Ldest** : liste des **adresseIP** de tous les destinataires du Message

Travail demandé :

1. Créer la classe **Message** ainsi que son constructeur à partir de ses 4 composants définis en dessus
2. Ecrire une méthode **__str__** qui construit pour un message donné, une chaîne de caractères contenant l'adresseIP de l'émission suivie par les adresseIP des destinataires, la date d'émission et le contenu du message séparés tous par un «-».
3. Ecrire une méthode **__contains__**, qui prend comme paramètre une adresseIP **adr** et qui retourne **True** si l'adresse **adr** est l'adresse IP d'un des destinataires du message et **False** sinon
4. Ecrire une méthode **Verif_destination**, à résultat booléen, permettant de vérifier si toutes les adresses des destinataires d'un message appartiennent au même réseau que celui de l'émetteur
5. Ecrire une méthode **__add__** permettant de fusionner deux Messages : en fait, qui à partir de deux messages émis par **un même utilisateur** construise un nouveau message défini par :
 - La date d'émission est la plus récente des deux dates d'émission de m1 et m2. On se propose avoir la méthode **Recente()** qui prend comme paramètre deux Messages et retourne la date la plus récente des deux dates d'émission de m1 et m2
 - Le contenu est la concaténation des deux contenus
 - La liste des destinataires résultat est l'union des deux listes des destinataires de m1 et m2 sans redondance.

Exemple : si **m1.Ldest=[adr1,adr2]** et **m2.Ldest=[adr1,adr3]** alors la liste résultante sera **[adr1,adr2,adr3]**

6. Ecrire une méthode : **Dupliquer()** qui à partir d'un Message, construit et retourne une liste **LM** de Messages, **dont chacun aura un seul destinataire parmi ceux du Message** .

NB : Les contenus et moments d'émissions seront identiques pour tous les Messages construits.

Problème 2

(9 points)

Dans cet exercice, on s'intéresse à la création des classes permettant de modéliser la gestion d'une pharmacie qui gère des clients et des produits.

1. Créer la classe **Produit** selon le principe suivant :

- 1.1.Définir le constructeur qui initialise les trois attributs suivants :

- **Ref** : une chaine, qui représente la référence du produit
- **Prix** : un réel, qui représente le prix du produit
- **Qte** : un entier, qui représente la quantité disponible en stock

- 1.2.Définir une méthode **Disponible** qui prend comme paramètre une quantité **qte**, de type entier, et retourne True si elle est disponible et False sinon

- 1.3.Définir une méthode **Reduction**, qui prend comme paramètre **P**, le pourcentage d'une remise, et retourne la valeur du prix du produit si on effectue une réduction de P%

2. Créer la classe **Medicament**, fille de la classe **Produit**, selon le principe suivant :

- 2.1.Le constructeur initialise les cinq attributs suivants :

- **Ref** : une chaine, qui représente la référence du produit
- **Prix** : un réel, qui représente le prix du produit
- **Qte** : un entier, qui représente la quantité disponible en stock
- **Ord** : un booléen indiquant si le médicament peut être délivré sans une ordonnance (True si l'ordonnance est obligatoire et False sinon)
- **CNAM** : un booléen indiquant si le médicament est remboursable par la CNAM (True s'il est remboursable et False sinon)

- 2.2.Définir une méthode **MedDisponible** qui prend comme paramètre une quantité **qte**, de type entier, et retourne True si elle est disponible et False sinon

- 2.3. Définir une méthode **Montant** qui prend comme paramètre une quantité **qte**, et retourne le montant total à payer, sachant que si le médicament est remboursable par la CNAM on paie seulement 40% du prix du médicament.

NB : La quantité en stock doit être mise à jour.

3. Définir la classe **Paramedical**, qui hérite de la classe produit, représentant les produits paramédicaux

- 3.1.Le constructeur de cette classe initialise les quatre attributs suivants :

- **Ref** : une chaine, qui représente la référence du produit
- **Prix** : un réel, qui représente le prix du produit
- **Qte** : un entier, qui représente la quantité disponible en stock
- **Type** : une chaine, indiquant le type du produit (produit de beauté, cosmétique ou diététique)

3.2.Définir une méthode **ParaDisponible** qui prend comme paramètre une quantité **qte**, de type entier, et retourne True si elle est disponible et False sinon

3.3.Définir une méthode **MontantPromo** qui prend comme paramètre **P**, le pourcentage d'une promo, **qte** la quantité désirée et retourne le montant à payer si on effectue une réduction de P% .

NB :La quantité en stock doit être mise à jour.

4. Définir la classe **Pharmacie** selon le principe suivant :

4.1. Définir le constructeur de cette classe qui initialise les trois attributs :

- **LMed** : une liste de **Medicaments**
- **LPara** : une liste de **Paramedicals**
- **LClient** : une liste initialement vide contenant les noms des clients de la pharmacie

4.2.Définir une méthode **VenteMed** qui prend comme paramètre une ordonnance **Ord**, le nom d'un client **nomC** , calcule et retourne le montant total de l'ordonnance.

Sachant que **Ord** est une liste des listes tel que chaque sous liste est de la forme **[référence_Med, Nb_Boites]**

NB : on suppose que tous les médicaments prescrits sont disponibles

4.3.Définir une méthode **VentePara** qui prend comme paramètre **refP**, la référence d'un Paramedical , **qte** la quantité désirée et **P**, un réel, qui représente le pourcentage de la promo, calcule et retourne le montant à payer

Université de Carthage Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul Département Mathématique- Informatique	 المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2019/2020 Filière : SM/SP/ST Niveau d'étude : 2^{ème} année Semestre : 1 Nombre de pages : 4 Date : 14/12/2019 Durée : 2h
--	---	---

Alternative de correction Examen 1 Informatique

Problème 1 :(11 points)

Partie 1

1	<pre>class AdresseIP: DClasse={'A':[1,126],'B':[128,191],'C':[192,223]} def __init__(self,Nb0,Nb1,Nb2,Nb3): self.Nb0=Nb0 self.Nb1=Nb1 self.Nb2=Nb2 self.Nb3=Nb3</pre>	<u>1pt</u>
2	<pre>def Classe(self): for c,l in AdresseIP.DClasse.items(): if l[0]<=self.Nb0<=l[1]:return(c)</pre>	<u>1pt</u>
3	<pre>def AdrReseau(self): c=self.Classe() if c=='A':return(AdresseIP(self.Nb0,0,0,0)) elif c=='B': return(AdresseIP(self.Nb0,self.Nb1,0,0)) else: return(AdresseIP(self.Nb0,self.Nb1,self.Nb2,0))</pre>	<u>1pt</u>
4	<pre>def __eq__(self,l): return(self.Nb0==l.Nb0 and self.Nb1==l.Nb1 and self.Nb2==l.Nb2 and self.Nb3==l.Nb3)</pre>	<u>0.5 pt</u>
5	<pre>def MemeReseau(self,l): return(self.AdrReseau()==l.AdrReseau())</pre>	<u>0.5 pt</u>
6	<pre>def __str__(self): F='{ }.{ }.{ }.{ }' return(F.format(self.Nb0,self.Nb1,self.Nb2,self.Nb3))</pre>	<u>0.5 pt</u>

Partie 2

1	<pre>class Message: def __init__(self,e,d,c,L): self.Emetteur=e self.Date=d self.Contenu=c self.Ldest=L</pre>	<u>0.5 pt</u>
2	<pre>def __str__(self): L=[str(l) for l in self.Ldest] F='{ }-{ }-{ }-{ }' return(F.format(str(self.Emetteur),'-'.join(L),self.Contenu,self.Date))</pre>	<u>1 pt</u>
3	<pre>def __contains__(self,adr):</pre>	<u>1 pt</u>

	<pre> for adrIP in self.Ldest: if adrIP==adr: return(True) return(False) </pre>	
4	<pre> def verifDestination(self): for adr in self.Ldest: if not adr.MemeReseau(self.Emetteur): return(False) return(True) </pre>	<u>1 pt</u>
5	<pre> def __add__(self,M): from copy import copy date=self.Recente(M) cont=self.Contenu+M.Contenu L=copy(self.Ldest) for adr in M.Ldest: if not adr in self: L.append(adr) return(Message(self.Emetteur,date,cont,L)) </pre>	<u>2pt</u>
6	<pre> def Dupliquer(self): L=[] for adr in self.Ldest: L.append(Message(self.Emetteur,self.Date,self.Contenu, [adr])) return(L) </pre>	<u>1pt</u>

Problème 2(9 points)

1. La classe Produit

1.1	<pre> class Produit: def __init__(self,r,p,q): self.Ref=r self.Prix=p self.Qte=q </pre>	<u>0.5pt</u>
1.2	<pre> def Disponible(self,qte): return(self.Qte>=qte) </pre>	<u>0.5pt</u>
1.3	<pre> def Reduction(self,p): return(self.Prix-self.Prix*p/100) </pre>	<u>0.5pt</u>

2. La classe Medicament

2.1	<pre> class Medicament(Produit): def __init__(self,r,p,q,o,cnam): Produit.__init__(self,r,p,q) self.Ord=o self.CNAM=cnam </pre>	<u>0.5pt</u>
2.2	<pre> def MedDisponible(self,qte): return(Produit.Disponible(self,qte)) </pre>	<u>0.5pt</u>
2.3	<pre> def Montant(self,qte): if self.MedDisponible(qte): self.Qte-=qte </pre>	<u>1pt</u>

	<pre> if self.CNAM: return(qte*Produit.Reduction(self,60)) else: return(qte*self.prix) </pre>	
--	---	--

3. La classe Paramédical

3.1	<pre> class Paramedical(Produit): def __init__(self,r,p,q,t): Produit.__init__(self,r,p,q) self.Type=t </pre>	<u>0.5pt</u>
3.2	<pre> def ParaDisponible(self,qte): return(Produit.Disponible(self,qte)) </pre>	<u>0.5pt</u>
3.3	<pre> def MontantPromo(self,qte,P): if self.ParaDisponible(qte): self.Qte-=qte return(qte*Produit.Reduction(self,P)) </pre>	<u>1pt</u>

4. La classe Pharmacie

4.1	<pre> class Pharmacie : def __init__(self,LM,LP): self.LMed=LM self.LPara=LP self.LClient=[] </pre>	<u>0.5pt</u>
4.2	<pre> def VenteMed(self,Ord,nomC): if nomC not in self.LClient: self.LClient.append(nomC) Montant=0 for e in Ord: for M in self.LMed: if M.Ref==e[0]: Montant+=M.Montant(e[1]) break return(Montant) </pre>	<u>2pts</u>
4.3	<pre> def VentePara(self,refP,qte,P): for Para in self.LPara: if Para.Ref==refP: return(Para.MontantPromo(qte,P)) </pre>	<u>1pt</u>



Epreuve d'informatique

1^{er} semestre AU : 2018 – 2019

Date : Novembre 2018

;

Durée : 1H

;

Nombre de page : 3

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve.

Le langage de programmation sera obligatoirement Python.

*

* *

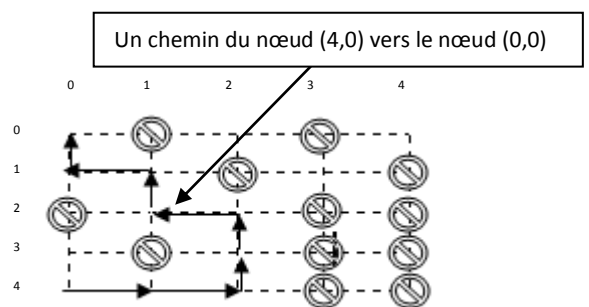
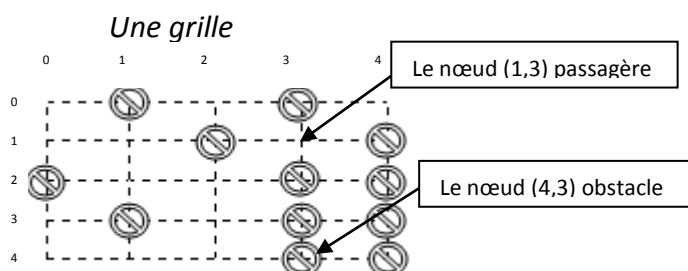
Implémentation. Dans ce sujet, nous adopterons la syntaxe du langage Python. On rappelle qu'en Python, il importe de bien respecter les indentations car elles permettent de définir des blocs.

Problème (Le labyrinthe)

Sur une grille discrète $[0, n-1] \times [0, n-1]$, on considère les n^2 nœuds $c = (i, j)$. Deux nœuds de la grille sont voisins s'il existe une arête horizontale ou verticale les reliant. Un chemin entre deux nœuds c_1 et c_2 est la suite de nœuds successivement voisins $c_1, c_i, c_j, \dots, c_k, c_2$, d'extrémités c_1 et c_2 .

Un labyrinthe est tel que, pour une paire de nœuds de la grille, il existe un chemin entre ces nœuds sans passer par les nœuds obstacles . Le labyrinthe d'entrée c_1 et de sortie c_2 sera noté ici le chemin :
 $[c_1, c_i] \rightarrow [c_i, c_j] \rightarrow [c_j, c_k] \rightarrow [c_k, c_r] \rightarrow [c_r, c_p] \rightarrow [c_p, c_2]$

Voici un exemple de labyrinthe de dimension 5×5 , et un chemin qui entre du coin inférieur gauche $(4, 0)$ et sort du coin supérieur gauche $(0, 0)$.



Chemin =

$[(4, 0), (4, 1)] \rightarrow [(4, 1), (4, 2)] \rightarrow [(4, 2), (3, 2)] \rightarrow [(3, 2), (2, 2)] \rightarrow [(2, 2), (2, 1)] \rightarrow [(2, 1), (1, 1)] \rightarrow [(1, 1), (1, 0)] \rightarrow [(1, 0), (0, 0)]$

L'objectif de notre problème consiste, à partir d'une grille, trouver un chemin entre le nœud de départ et le nœud d'arrivée.

L'approche suivante profite de la structure de pile. On se donne :

- un **nœud** : un tuple (i,j) indiquant les coordonnées du nœud dans la grille.
- une **grilleDesVisites** : une liste de liste de booléens (1:True, 0:False), pour identifier les nœuds passagère : 0, les nœuds obstacles et déjà visités : 1 ,
- une pile **p** des nœuds non encore visités,
- une pile **laby** qui stocke le chemin du labyrinthe en construction.

On part du nœud d'entrée, par exemple (n-1, 0), que l'on empile sur **p** et que l'on marque dans **grilleDesVisites**.

Tant que la pile p n'est pas vide :

- i) on dépile le "nœud-sommet" c de la pile p,
- ii) on identifie ses nœuds voisins non encore visités et si il y en a au moins un :
 - on en choisit un voisin aléatoirement : s,
 - on empile le chemin [c, s] sur **laby**,
 - on empile c puis s sur p,
 - on marque s dans **grilleDesVisites** et on reprend.
 - Si s est le nœud d'arrivée on s'arrête.

Les piles à traiter sont des piles bornées dont le premier élément indique la taille. On pourra utiliser les fonctions suivantes du **module pile** :

- Créer_pile_vide(c) : permet de créer une pile vide à capacité fini c
- Empiler(p, v) : ajouter v au sommet de la pile p
- Dépiler(p) : retire le sommet de la pile et retourne l'élément dépilé
- Sommet(p) : retourne le sommet de la pile.
- Est_vide(p) : vérifie si p est vide
- Taille(p) : renvoie la taille de p

Travail demandé :

Les questions suivantes vous accompagnent dans la mise en œuvre de cet algorithme. La seule difficulté est **d'éviter de sortir de la grille**.

Question1.

Les grilles de labyrinthe sont stockées dans des fichiers textes dont les lignes sont une suite de 0 et de 1 séparée par des virgules représentant les nœuds de grille (0 : un nœud passagère, 1 : un nœud obstacle).

Ecrire la fonction **upload_grille(fichier)** qui lit le fichier ligne par ligne et construit la grille des visites sous la forme d'une liste de liste.

Exemple :

Soit le fichier grille01.txt suivant :

```
>>> grilleDesVisites = upload_grille('grille01.txt')
>>> print(grilleDesVisites)
[[0, 1, 0, 1, 0], [0, 0, 1, 0, 1], [1, 0, 0, 1, 1], [0, 1, 0, 1, 1], [0, 0, 0, 1, 1]]
```

grille01.txt

0	1	0	1	0
0	0	1	0	1
1	0	0	1	1
0	1	0	1	1
0	0	0	1	1

Question2.

Ecrire la fonction **initialiser_piles(n)** qui prend en paramètre un entier n et qui crée et renvoie les deux piles bornées vides p et laby de capacité n^2 .

Question3.

Ecrire la fonction **visiter(noeud , grilleDesVisites)** qui marque le nœud comme étant déjà visité dans la grille des visites.

Exemple :

```
>>> visiter( (3,2) , grilleDesVisites )
>>> print(grilleDesVisites)
[[0, 1, 0, 1, 0], [0, 0, 1, 0, 1], [1, 0, 1, 1, 1], [0, 1, 1, 1, 1], [0, 0, 0, 1, 1]]
```

Question 4.

Ecrire la fonction **dejaVisite(noeud, grilleDesVisites)** qui renvoie l'état de visite du nœud donné, si le nœud est dans la grille. Sinon (si le nœud est hors grille) la fonction renvoie 1.

Question 5.

Ecrire la fonction **voisins (nœud , grilleDesVisites)** qui renvoie la liste des nœuds voisins non encore visités du nœud donné (passé en paramètre).

Question6.

Ecrire la fonction **choisirVoisin(nœud , grilleDesVisites)** qui choisit et renvoie un nœud voisin au hasard non encore visité du nœud donné. Cette fonction traite l'exception dans le cas où le nœud ne possède pas de voisins.

NB : Utiliser la fonction `choice(L)` du module `random` qui renvoie au hasard un élément de la liste L.

Question 7.

Ecrire la fonction **labyrinthe(grilleDesVisites, depart , arrivee)** qui prend en paramètre la grille des visites, le nœud de départ et le nœud d'arrivée. Cette fonction calcule et renvoie la pile **laby** selon l'algorithme décrit précédemment.

Exemple :

```
>>> laby = labyrinthe(grilleDesVisites, (4,0) , (0,0) )
>>> print(laby)
[8, [(4, 0), (4, 1)], [(4, 1), (4, 2)], [(4, 2), (3, 2)], [(3, 2), (2, 2)],
[(2, 2), (2, 1)], [(2, 1), (1, 1)], [(1, 1), (1, 0)], [(1, 0), (0, 0)],
None, None, None, None, None, None, None, None, None, None, None, None]
```

None
...
None
[(1, 0), (0, 0)]
[(1, 1), (1, 0)]
[(2, 1), (1, 1)]
[(2, 2), (2, 1)]
[(3, 2), (2, 2)]
[(4, 2), (3, 2)]
[(4, 1), (4, 2)]
[(4, 0), (4, 1)]
8

Pile borne : laby

Question 8.

Ecrire la fonction **tracer_chemin(laby)** qui parcourt la pile laby et renvoie le chemin du labyrinthe en chaîne de caractères sous la forme ' $[c_1, c_i] \rightarrow [c_i, c_j] \rightarrow [c_j, c_k] \rightarrow [c_k, c_r] \rightarrow [c_r, c_p] \rightarrow [c_p, c_2]$ '.

Exemple :

```
>>> chemin = tracer_chemin(laby)
>>> print(chemin)
' [(4, 0), (4, 1)]->[(4, 1), (4, 2)]->[(4, 2), (3, 2)]->[(3, 2), (2, 2)]->[(2, 2), (2, 1)]->[(2, 1), (1, 1)]
->[(1, 1), (1, 0)]->[(1, 0), (0, 0)] '
```

En déduire la **complexité** au pire des cas de la fonction **tracer_chemin** en fonction de n.

Corrige DS1 2018_2019 (labyrinthe)

November 1, 2018

1 Question 1 (2 pts)

```
In [158]: def upload_grille(fichier):  
    f = open(fichier, 'r')  
    grille = list()  
    for ligne in f :  
        L = ligne.split(',')  
        L = [int(e) for e in L]  
        grille.append(L)  
    f.close()  
    return grille
```

2 Question 2 (2 pts)

```
In [159]: from pile import *  
    def initialiser_piles(n):  
        p = creer_pile_vide(n*n)  
        laby = creer_pile_vide(n*n)  
        return p , laby
```

3 Question 3 (1pt)

```
In [160]: def visiter(noeud , grilleDesVisites):  
    (i,j) = noeud  
    grilleDesVisites[i][j] = 1
```

4 Question 4 (2 pts)

```
In [161]: def dejaVisite(noeud, grilleDesVisites):  
    (i,j) = noeud  
    if i < 0 or i >= len(grilleDesVisites) or j < 0 or j >= len(grilleDesVisites):  
        return True  
    return grilleDesVisites[i][j]
```

5 Question 5 (2 pts)

```
In [162]: def voisines(noeud , grilleDesVisites):
            V = list()
            (i,j) = noeud
            v1 = (i-1,j)
            if not(dejaVisite(v1,grilleDesVisites)):
                V.append(v1)
            v2 = (i+1,j)
            if not(dejaVisite(v2,grilleDesVisites)):
                V.append(v2)
            v3 = (i,j-1)
            if not(dejaVisite(v3,grilleDesVisites)):
                V.append(v3)
            v4 = (i,j+1)
            if not(dejaVisite(v4,grilleDesVisites)):
                V.append(v4)
            return V
```

6 Question 6 (3 pts)

```
In [163]: import random
            def choisirVoisin(noeud, grilleDesVisites):
                V = voisines(noeud,grilleDesVisites)
                try :
                    assert(len(V) > 0 )
                    noeudVoisin = random.choice(V)
                    return noeudVoisin
                except:
                    return None
```

6.1 Question 7 (4 pts)

```
In [164]: def labyrinthe(grille, depart , arrivee):
            n = len(grille)
            p , laby = initialiser_piles(n*n)
            empile(p,depart)
            visiter(depart,grille)
            while not(est_vide(p)):
                c = depiler(p)
                s = choisirVoisin(c,grille)
                if s != None :
                    empile(laby,[c,s])
                    empile(p,c)
                    empile(p,s)
                    visiter(s,grille)
```

```

        if s == arrivee:
            return laby
    return None

```

7 Question 8 (3 pts)

In [165]: `def tracer_chemin(laby):`

```

    chemin = ""
    try :
        assert(laby != None)
        aux = creer_pile_vide(laby[0])
        while not(est_vide(laby)):
            empile(aux,depiler(laby))
        while not(est_vide(aux)):
            chemin += "{}->".format(depiler(aux))
    except :
        print("pas de chemin !!!")
    return chemin

```

8 Complexite : (1 pt)

Dans le pire des cas la pile laby est de taille n^2 , la fonction tracer_chemin parcourt 2 fois la pile laby, donc sa complexite est de l'ordre de $O(n^2)$.

EXAMEN DE FIN DE TRIMESTRE 2**Matière : INFORMATIQUE****Classes : 2^{ème} Année Durée : 1h30**

Préparation : MP, PC, PT et PB

DOCUMENTS NON AUTORISES**EXERCICE 1**

Une image numérique peut être représentée par une matrice dans laquelle chaque case représente un pixel (unité minimale adressable par le contrôleur vidéo et supposé de couleur uniforme). Le *format* de l'image (png, jpeg, etc) désigne la manière dont cette matrice est représentée en mémoire.

- Pour une image en noir et blanc, chaque pixel ne peut prendre que deux valeurs, ce qui permet de l'encoder sur un seul bit ;
- pour une image en niveau de gris, chaque teinte de gris (en passant du noir au blanc) est encodée sur un octet, ce qui permet de représenter $2^8 = 256$ nuances de gris;
- pour une image en couleur, chaque pixel est encodé sur trois octets, chaque octet représentant une nuance de couleur. Par exemple, dans le cas du format RGB (*Red*, *Green*, *Blue*), le premier octet dose la quantité de rouge, le second la quantité de vert et le troisième la quantité de bleu pour une synthèse additive ;
- enfin, les pixels d'une image en couleur peuvent être encodés par quatre octets au format RGBA (A pour *alpha*), le quatrième octet codant la transparence du pixel.

Manipulation d'image en PYTHON

Durant cette EPREUVE nous aurons besoin de trois modules : NUMPY OU SCIPY pour la manipulation des matrices, MATPLOTLIB.PYLOT pour la visualisation des images et enfin IMAGEIO pour la conversion image/matrice. Nous aurons besoin aussi des fonctions suivantes:

- La fonction `io.imread(source)` permet de convertir une image au format courant (png, jpeg, etc) en un tableau NUMPY. Elle prend en argument une chaîne de caractère décrivant un fichier présent sur votre ordinateur ou l'adresse http d'une image présente sur le net.
- La fonction `io.imwrite(cible, tab)` permet au contraire de convertir un tableau NUMPY (l'argument *tab*) en un fichier image (décrit par la chaîne de caractères *cible*).
- La fonction `plt.imshow(tab)` permet de visualiser l'image décrite par un tableau. À noter, pour visualiser une image en niveau de gris il faut préciser l'échelle chromatique à utiliser, ici : `plt.imshow(tab, cmap= 'gray ')`.

Question 0.

Ecrire les instructions permettant :

- d'importer les modules nécessaires pour la manipulation des matrices
- de récupérer l'image 'picasso.png' qui nous servira de modèle, la visualiser puis l'enregistrer dans le répertoire courant.

Ainsi définie `im` est tableau NUMPY dont les éléments sont de type `np.uint8` (entiers non signés représentés sur 8 bits, autrement dit un entier compris entre 0 et 255). La méthode `shape` donne les dimensions de ce tableau : `im.shape = (n, p)` pour une image en niveau de gris, `im.shape = (n, p, 3)` pour une image couleur, `im.shape = (n, p, 4)` pour une image avec composante alpha. L'entier n est le nombre de lignes de l'image et p le nombre de colonnes, le pixel de coordonnées (0,0) est situé en haut à gauche de l'image.

Remarque. En réalité, le format de l'objet importé par la commande `io.imread` est plus complexe qu'un simple tableau NUMPY (il contient des informations complémentaires sur le contenu de l'image) et la manipulation de ces images s'en trouve ralentie. Pour accélérer la vitesse d'exécution des fonctions que vous aurez à écrire, vous aurez intérêt à transformer ces dernières en simples tableaux NUMPY en rajoutant au code précédent la commande :

```
im = np.array(im)
```

Question 1.

Rédiger une fonction **symetrie** qui prend en argument une image et retourne une nouvelle image, obtenue par symétrie autour d'un axe vertical passant par le milieu de l'image.

Indication : la fonction `np.zeros_like(m)` crée un tableau nul de même type et de mêmes dimensions que le tableau `m`.

Question 2.

Rédiger une fonction **rotation** qui prend en argument une image et retourne une nouvelle image, obtenue par rotation d'un angle $n/2$ autour du centre de l'image.

Indication : la fonction `np.zeros((n, p, s), dtype=np.uint8)` crée un tableau vide de dimensions $n \times p \times s$ dont les éléments sont de type `np.uint8`.

Question 3.

Rédiger une fonction **negatif** qui prend en argument une image et retourne son négatif, c'est-à-dire l'image dans laquelle chaque composante de couleur de chaque pixel est remplacée par sa valeur complémentaire dans l'intervalle $[0,255]$.

Question 4. Conversion en niveau de gris

Pour convertir une image couleur en niveau de gris, un pixel représenté par ses composantes (r,g,b) doit être remplacé par un pixel à une seule composante $y \in [0,255]$ appelée *luminance*. Cette quantité, qui traduit la sensation visuelle de luminosité, dépend de manière inégale des trois composantes RGB. La formule communément recommandée pour cette conversion est la suivante :

$$y = 0,2126 r + 0,7152 g + 0,0722 b \text{ (} y \text{ étant arrondi à l'entier le plus proche)}$$

(pour un œil humain le vert paraît plus lumineux que le rouge, lui-même plus lumineux que le bleu).

Rédiger une fonction **niveaudegris** qui prend en argument une image couleur et retourne une nouvelle image convertie en niveau de gris.

Traitement d'image

La plupart des filtres de traitement d'images utilisent une convolution par un *masque* pour réaliser une modification. Ces masques sont des matrices de petites tailles, en général 3×3 (taille que nous allons considérer par la suite) ou 5×5 :

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

Le filtre associé à ce masque transforme la matrice $M = (m_{ij})$, associée à une certaine image, en la matrice $M \otimes C = (m^*_{ij})$ par une opération appelée *produit de convolution* et définie par la relation : $\forall i, j$,

$$m^*_{ij} = c_{11}m_{i-1,j-1} + c_{12}m_{i-1,j} + c_{13}m_{i-1,j+1} + c_{21}m_{i,j-1} + c_{22}m_{i,j} + c_{23}m_{i,j+1} + c_{31}m_{i+1,j-1} + c_{32}m_{i+1,j} + c_{33}m_{i+1,j+1}$$

Dans le cas d'une image en couleur, on peut choisir d'appliquer le même masque à chacune des trois composantes RGB de l'image ou leur appliquer des masques différenciés, en fonction de l'effet désiré. Dans la suite de ce sujet, et dans un but simplificateur, nous appliquerons le même masque à chacune des trois composantes de couleur.

Remarque. Lorsque le pixel initial est sur un bord, une partie de la convolution porte en dehors des limites de l'image. Là encore, nous conviendrons pour simplifier de laisser inchangés ces pixels.

Question 5.

Rédiger une fonction **convolution** qui prend en arguments deux matrices M (associée à une image) et C et retourne la matrice $M \otimes C$. Les éléments de la matrice C seront *a priori* de type *float* tandis que ceux des matrices M et $M \otimes C$ seront de type *np.uint8*. Il conviendra donc, lorsque $m^*_{ij} < 0$ ou $m^*_{ij} > 255$ de remplacer les valeurs calculées par respectivement 0 et 255. Ne pas oublier non plus que les pixels d'une image en couleur possèdent trois composantes et que chacune de ces composantes doit être traitée.

Exemples de masques

Lissage Pour obtenir un effet de lissage on utilise le masque C_1 : chaque pixel est remplacé par la moyenne de lui-même et de ses huit voisins ; pour cette raison on parle de filtre *moyenneur*.

Augmentation du contraste Au contraire, pour augmenter le contraste on utilise le masque C_2 .

Repoussage Enfin, le masque C_3 donne un effet de relief à l'image, appelé *repoussage*.

$$C_1 = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad C_2 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad C_3 = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

FIGURE 1 – Trois filtres classiques en traitement d'images.

Question 6.

Rédiger trois fonctions **lissage**, **contraste** et **repoussage** qui prennent toutes trois en argument une image et retournent une nouvelle image obtenue en appliquant respectivement un effet de lissage, d'augmentation de contraste et de repoussage, et observer le résultat sur l'image test.

EXERCICE 2

L'objectif de l'exercice est l'utilisation de la méthode des moindres carrés pour approximer un polynôme à partir d'observations. La spécificité de cette méthode est de minimiser la somme des distances entre un polynôme g approximant et n points expérimentaux.

Etant donné une observation de n points distincts $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, on cherche à approcher cette observation, au sens des moindres carrés, par un polynôme $g(x) = a_0 + a_1x + a_2x^2 + \dots + a_mx^m$ de degré m égal

à $n-1$, donnant une meilleure approximation des valeurs y_i . Ceci revient à minimiser la distance qui sépare un point expérimental (x_i, y_i) du point approximant $(x_i, g(x_i))$.

Le polynôme donnant la meilleure approximation est celui qui minimise la somme S , des écarts entre les y_i et les $g(x_i)$, donnée par la formule suivante :

$$S(a_0, a_1, a_2, \dots, a_m) = \sum_{i=1}^n |y_i - (a_0 + a_1 x_i + a_2 x_i^2 + \dots + a_m x_i^m)|$$

Afin d'identifier la distance moyenne minimale, on cherche l'ensemble des valeurs des paramètres a_i minimisant cette somme.

Ce qui conduit à résoudre le système linéaire suivant :

$$\begin{bmatrix} \sum_{i=1}^n x_i^0 & \sum_{i=1}^n x_i^1 & \dots & \sum_{i=1}^n x_i^m \\ \sum_{i=1}^n x_i^1 & \sum_{i=1}^n x_i^2 & \dots & \sum_{i=1}^n x_i^{m+1} \\ \vdots & \vdots & \dots & \vdots \\ \sum_{i=1}^n x_i^{m-1} & \sum_{i=1}^n x_i^m & \dots & \sum_{i=1}^n x_i^{2m-1} \\ \sum_{i=1}^n x_i^m & \sum_{i=1}^n x_i^{m+1} & \dots & \sum_{i=1}^n x_i^{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n x_i^0 y_i \\ \sum_{i=1}^n x_i^1 y_i \\ \vdots \\ \sum_{i=1}^n x_i^{m-1} y_i \\ \sum_{i=1}^n x_i^m y_i \end{bmatrix}$$

En posant $U_k = \sum_{i=1}^n x_i^k$ et $v_k = \sum_{i=1}^n x_i^k y_i$, le système s'écrit alors :

$$\begin{bmatrix} U_0 & U_1 & \dots & U_m \\ U_1 & U_2 & \dots & U_{m+1} \\ \vdots & \vdots & \dots & \vdots \\ U_{m-1} & U_m & \dots & U_{2m-1} \\ U_m & U_{m+1} & \dots & U_{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{m-1} \\ v_m \end{bmatrix} \Rightarrow U \cdot a = v$$

Travail demandé :

Dans la suite, on suppose que les n points représentant une observation, sont stockés dans une liste de tuples Lp , où chaque tuple représente les coordonnées d'un point.

Le travail demandé consiste à déterminer les coefficients a_i du polynôme d'interpolation $g(x)$.

1. Ecrire une fonction python, nommée **puiss**, qui, pour un réel x et un entier p , crée et retourne la liste $[x^0, x^1, \dots, x^{2p}]$.
2. Ecrire une fonction python, nommée **list_puiss**, qui, à partir d'une liste de réels L et d'un entier p , crée et retourne une liste de listes contenant, pour chaque réel r de L , une liste $[r^0, r^1, \dots, r^{2p}]$.
3. Ecrire une fonction python, nommée **calcul_mat**, qui, à partir de la liste de points Lp , crée et retourne la matrice U .
4. Ecrire une fonction python, nommée **calcul_vect**, qui à partir de la liste de points Lp , crée et retourne le vecteur v .

Correction examen 2A S2 2018

EXRCICE1: 2pts par question

Question 0.

```
import numpy as np
import matplotlib.pyplot as plt
import imageio as io
im = io.imread('picasso.png')
plt.imshow(im)
plt.show()
io.imwrite('./picasso.png', im)
```

Question 1.

```
def symetrie(im):
    n, p, _ = im.shape
    im_sym = np.zeros_like(im)
    for i in range(n):
        for j in range(p):
            im_sym[i, j] = im[i, p-1-j]
    return im_sym
```

Question 2.

```
def rotation(im):
    n, p, s = im.shape
    im_rot = np.zeros((n, p, s), dtype=np.uint8)
    for i in range(n):
        for j in range(p):
            im_rot[p-1-j, i] = im[i, j]
    return im_rot
```

Question 3.

```
def negatif(im):
    n, p, _ = im.shape
    im_neg = np.zeros_like(im)
    for i in range(n):
        for j in range(p):
            im_neg[i, j] = [255, 255, 255] - im[i, j]
    return im_neg
```

Question 4.

```
def luminance(p):
    return np.round(0.2126*p[0] + 0.7152*p[1] + 0.0722*p[2])

def niveaudegris(im):
    n, p, _ = im.shape
    im_lum = np.zeros((n, p), dtype=np.uint8)
    for i in range(n):
        for j in range(p):
            im_lum[i, j] = luminance(im[i, j])
    return im_lum
```

Question 5.

```
def convolution(m, c):
    n, p, q = m.shape
    mm = np.zeros_like(m)
    for i in range(1, n-1):
        for j in range(1, p-1):
            s = 0
            for u in range(3):
                for v in range(3):
                    s += c[u, v] * m[i-1+u, j-1+v]
            for k in range(q):
                if s[k] < 0:
                    mm[i, j, k] = 0
                elif s[k] > 255:
                    mm[i, j, k] = 255
                else:
                    mm[i, j, k] = s[k]
    return mm
```

Question 6.

```
def lissage(im):
    c = 1/9*np.ones((3, 3), dtype=float)
    return convolution(im, c)

def contraste(im):
    c = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]], dtype=float)
    return convolution(im, c)

def repoussage(im):
    c = np.array([[-2, -1, 0], [-1, 1, 1], [0, 1, 2]], dtype=float)
    return convolution(im, c)
```

EXRCICE 2: 2+2+2+2 points

Question 1 (2pt)

```
def puiss(x,p):
    return [x**i for i in range(2*p+1)]
```

Question 2 (2 pt)

```
def list_puiss(L,p):
    return [puiss(L[i],p) for i in range(len(L))]
```

Question 3 (2 pts)

```
def calcul_mat(Lp):
    n=len(Lp)
    m=n-1
    M=np.ones((n,n))
    Lx=[Lp[i][0] for i in range(n)]
    Lpx=list_puiss(Lx,n-1)
    Lpxs=[sum([Lpx[i][j] for i in range(n)]) for j in range(2*m+1) ]
    k=0
    print(Lpxs)
    for i in range(n):
        L=Lpxs[k:n+k]
        for j in range(n):
            M[i,j]=L[j]
        k+=1
```

```
#M[:,j]=Lpxs[k:n]
```

```
#k+=1
```

```
return M
```

Question 4 (2 pts)

```
def calcul_vect(Lp):
```

```
    n=len(Lp)
```

```
    m=n-1
```

```
    V=np.ones((n))
```

```
    Lx=[Lp[i][0] for i in range(n)]
```

```
    Ly=[Lp[i][1] for i in range(n)]
```

```
    Lpx=list_puiss(Lx,n-1)
```

```
    print(Lx,Lpx,Ly)
```

```
    Lsxy=[sum([Lpx[i][j]*Ly[i] for i in range(n)]) for j in range(m+1) ]
```

```
    print(Lsxy)
```

```
    for i in range(n):
```

```
        V[i]=Lsxy[i]
```

```
    return V
```



Epreuve d'informatique

1^{er} semestre AU : 2019 – 2020

Date : 6 Janvier 2020 ; Durée : 2H ; Nombre de page : 4

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve.

*Le langage de programmation sera obligatoirement **Python**.*

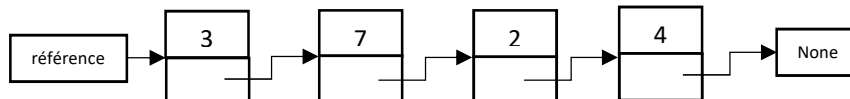
*

* *

Implémentation. Dans ce sujet, nous adopterons la syntaxe du langage Python. On rappelle qu'en Python, il importe de bien respecter les indentations car elles permettent de définir des blocs.

Partie I : Notion de liste chaînée

Une liste chaînée est une structure de données qui permet de stocker une séquence d'objets. La séquence d'entiers 3, 7, 2, 4 peut être illustrée par la figure suivante :



Où référence à une liste est une référence au 1^{er} élément qui constitue la tête. Le dernier élément (queue) pointe vers un objet vide (None).

En considérant que la liste chaînée la plus simple est la liste avec un seul élément, on peut donner une définition récursive aux listes chaînées :

- Soit le successeur de la tête est None
- Soit le successeur de la tête est une autre liste chaînée

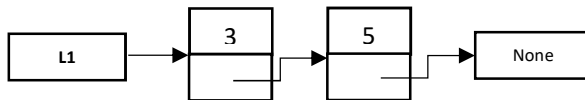
La condition générale d'arrêt de récursivité est un successeur None.

Dans ce problème, on représente une liste chaînée par la classe **Element** dont les objets sont caractérisés par les attributs suivants :

- data : valeur de l'élément, de type **int** ou **float**
- next : un pointeur vers son successeur, de type **Element**

```
class Element :  
    def __init__(self , dat, nex = None ):  
        self.data = dat  
        self.next = nex
```

Par exemple, soit la liste chaînée suivante :



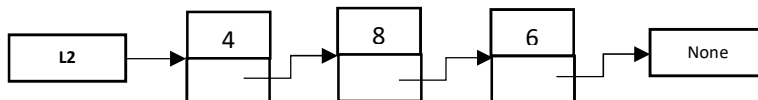
Cette liste est définie en Python comme suit :

```
>>> L0 = Element(5)
>>> L1 = Element(3, L0)
>>> print(L1.data)
3
>>> print(L1.next.data)
5
```

Travail demandé : (Classe *Element*)

Q1 : Ajouter la méthode `__str__` qui renvoie une chaîne qui représente une liste chaînée.

Par exemple, soit la liste suivante L2 :



```
>>> print(L2)
'4 --> 8 --> 6 -->'
```

Q2 : Ajouter la méthode `__len__` qui renvoie la longueur de la liste chaînée.

```
>>> print(len(L2))
3
```

Q3 : Ajouter la méthode récursive `sum` qui renvoie la somme des valeurs des éléments de la liste chaînée.

```
>>> print(L2.sum())
18
```

Q4 : Ajouter la méthode récursive `__contains__` qui prend en paramètre une valeur *v* et qui renvoie **True** si *v* existe dans les valeurs des éléments de la liste chaînée, **False** sinon.

```
>>> print(8 in L2)
True
```

Q5 : Ajouter la méthode `__getitem__` qui prend en paramètre un indice *i* et qui renvoie l'objet *Element* à la position *i* dans la liste chaînée.

```
>>> e = L2[1]
>>> print(e.data)
8
```

Q6 : Ajouter la méthode `push` qui prend en paramètre un indice *i* et une valeur *v* et qui ajoute un objet *Element* de valeur *v* à la position *i* dans la liste chaînée, tels que $1 \leq i < n$ (*n* étant la longueur de la liste chaînée).

```
>>> L2.push(2,5)
>>> print(L2)
'4 --> 8 --> 5 --> 6 -->'
```

Q7 : Ajouter la méthode **pop** qui prend en paramètre un indice **i** et qui supprime et renvoie l'objet **Element** à la position **i** dans la liste chaînée, tels que $1 \leq i < n$ (**n** étant la longueur de la liste chaînée).

```
>>> print(L2)
'4 --> 8 --> 6 -->'
>>> x = L2.pop(1)
>>> print(x.data)
8
>>> print(L2)
'4 --> 6 -->'
```

Q8 : Ajouter la méthode **__add__** qui prend en paramètre un objet **Element other** et qui étend la liste chaînée **self** par la liste chaînée **other** dans une nouvelle liste chaînée. Cette méthode renvoie la liste étendue.

```
>>> print(L2)
'4 --> 8 --> 6 -->'
>>> L3 = L2 + L2
>>> print(L3)
'4 --> 8 --> 6 --> 4 --> 8 --> 6 -->'
```

Partie II : Notion de Pile Bornée

Une pile bornée est une structure de donnée de type LIFO (Last In First Out) qui permet de stocker une séquence d'objet dans la limite de sa capacité.

Dans ce problème une pile a les caractéristiques suivantes :

- C'est une liste chaînée avec la valeur du premier élément vaut le nombre d'éléments présents dans la pile.
- Elle est dotée d'un attribut **capacity** qui limite le nombre d'éléments stockés dans la pile.
- Elle gère deux opérations :
 - **push** : empiler un élément à la queue de la liste chaînée
 - **pop** : dépiler la queue de la liste chaînée

On représente cette structure Pile par la classe **Stack** :

```
class Stack(Element):
    def __init__(self, cap = 100):
        Element.__init__(self, 0)      # ou super().__init__(0)
        self.capacity = cap
```

Travail demandé : (Classe Stack)

Q9 : Ajouter la méthode **is_empty** qui renvoie **True** si la pile est vide, **False** sinon.

```
>>> P1 = Stack(5)
>>> print(P1)
'0 -->'
>>> print(P1.is_empty())
True
```

Q10 : Ajouter la méthode **is_full** qui renvoie **True** si la pile est pleine, **False** sinon

```
>>> print(P1.is_full())  
False
```

Q11 : Redéfinir la méthode **push** qui prend en paramètre une valeur **v** et qui empile un objet **Element** de valeur **v**. Cette méthode traite l'exception dans le cas où la pile est pleine.

```
>>> P1.push(9)  
>>> P1.push(7)  
>>> print(P1)  
'2 -->9 --> 7 -->'  
>>> # 2 désigne le nombre d'éléments empilés
```

Q12 : Redéfinir la méthode **pop** qui dépile la queue de la liste chaînée. Cette méthode traite l'exception dans le cas où la pile est vide.

```
>>> print(P1)  
'2 -->9 --> 7 -->'  
>>> x = P1.pop()  
>>> print(x.data)  
7  
>>> print(P1)  
'1 -->9 -->'
```

Partie III : Voyageur de commerce

On souhaite dans cette partie aider le voyageur de commerce à traverser quelques villes de la Tunisie le plus rapidement possible. On cherchera à implémenter quelques astuces qui permettront de construire un chemin acceptable.

La première étape consiste à représenter les villes sous forme d'objets de la classe **City**.

Travail demandé : (Classe **City**)

Q13 : Définir la classe **City** avec un constructeur qui prend en paramètres : le nom, la longitude et la latitude de la ville et qui initialise les attributs suivants : **name**, **long**, **lat** avec les paramètres correspondants.

Q14 : Ajouter la méthode **distance** qui prend en paramètre un autre objet **City** **other** et qui renvoie la distance euclidienne entre les deux villes.

L'objectif ensuite est de récupérer le chemin proposé dans un fichier texte où chaque ligne est de la forme : « **nom_ville :longitude :latitude** », et puis de calculer la distance parcourue en suivant ce chemin.

Par exemple soit le fichier texte nommé « cities.txt »

Travail demandé :

Q15 : Ecrire la fonction **loadCities** qui prend en paramètres le nom du fichier texte et qui renvoie une liste d'objet **City**.

Q16 : Ecrire la fonction **setWay** qui prend en paramètres le nom du fichier texte et qui remplit une pile chemin (objet **Stack**), dont chaque valeur de son élément est la distance entre la ville courante et la ville précédente dans la pile.

Q17 : Ecrire la fonction **weightWay** qui prend en paramètres le nom du fichier texte et qui renvoie la distance du chemin parcouru.

cities.txt	
File	Edit View Language
1	Sousse:10.63699:35.82539
2	Douz:9.038601:33.456535
3	Sidi Bouzid:9.48494:35.03823
4	Sfax:10.766163:34.747847
5	Gabes:10.097522:33.888077
6	Tozeur:8.122933:33.918534
7	

CORRIGE DE L EPREUVE INFORMATIQUE S1-2020 POUR 2 EME ANNEE MP-PC-PT

January 12, 2020

1 Barème: 100 pts

2 Partie I : Définition d'une liste chaînée (45 pts)

In [63]: `class Element :`

```
    def __init__(self , dat , nex = None ):
        self.data = dat
        self.next = nex
    #Q1 : 5 pts
    def __str__(self):
        ch = ""
        courant = self
        while courant is not None :
            ch += "{}-->".format(courant.data)
            courant = courant.next
        return ch
    #Q2 : 5 pts
    def __len__(self):
        l = 0
        courant = self
        while courant is not None :
            l += 1
            courant = courant.next
        return l
    #Q3 : 5 pts
    def sum (self):

        if self.next is None :
            return self.data
        else :
            return self.data + self.next.sum()
    #Q4 : 5 pts
    def __contains__(self , v):
        if self is None :
```

```

        return False
    elif v == self.data :
        return True
    else :
        return v in self.next
#Q5 : 5 pts
def __getitem__(self , i ) :
    assert i in range(0,len(self))
    courant = self
    for j in range(i):
        courant = courant.next
    return courant
#Q6 : 7.5 pts
def push(self , i , v) :

    assert i in range(1,len(self)+1)
    e = Element(v)

    if i == len(self):
        queue = self[i-1]
        queue.next = e
    else :
        precedent=self[i-1]
        suivant = self[i]
        precedent.next=e
        e.next=suivant

#Q7 : 7.5 pts
def pop(self , i) :

    assert i in range(1,len(self))
    courant = self
    if i == len(self)-1:
        queue = courant[i]
        courant[i-1].next = None
        return queue
    else :
        precedent=self[i-1]
        e = courant[i]
        suivant = self[i+1]
        precedent.next = suivant
        return e

#Q8 : 5 pts
def __add__(self , other) :

    eltEtendu = Element(self[0].data)

```

```

    for i in range(1, len(self)):
        eltEtendu.push(i, self[i].data)

    queue = eltEtendu[i]
    queue.next = other

    return eltEtendu

```

3 Partie II : Définition d'une Pile (25 pts)

```

In [67]: class Stack(Element):
    def __init__(self, cap = 100):
        super().__init__(0)
        self.capacity = cap
    #Q9 : 5 pts
    def is_empty(self):
        return self[0].data == 0
    #Q10 : 5 pts
    def is_full(self) :
        return len(self)-1 == self.capacity
    #Q11 : 7.5 pts
    def push(self, v):
        try :
            assert not self.is_full()
            n = len(self) # n = self[0].data
            super().push(n, v)
            self[0].data += 1
        except :
            print("Pile Pleine")
    #Q12 : 7.5 pts
    def pop(self):
        try :
            assert not self.is_empty()
            n = self[0].data
            mail = super().pop(n)
            self[0].data -= 1
            return mail.data
        except :
            print("Pile Vide")

```

4 Partie III : Voyageur de commerce (30 pts)

```

In [69]: import math
        class City :

```

```

#Q13 : 5 pts
def __init__(self, name, long , lat):
    self.name = name
    self.long = long
    self.lat = lat

#Q14 : 5 pts
def distance(self, other):
    X = self.long - other.long
    Y = self.lat - other.lat
    return math.sqrt(X**2 + Y**2)

```

```

In [70]: #Q15 : 7.5 pts
def loadCities(file_name):
    f = open(file_name , 'r')
    listVille = list()
    for ligne in f :
        ligne = ligne [: -1]
        L = ligne.split(':')
        vil = City(L[0] , float(L[1]) , float(L[2]))
        listVille.append(vil)
    f.close()
    return listVille

```

```

In [71]: #Q16 : 7.5 pts
def setWay(file_name):
    L = loadCities(file_name)
    chemin = Stack(len(L))
    chemin.push(0)
    for i in range(1,len(L)):
        dist = L[i].distance(L[i-1])
        chemin.push(dist)
    print(chemin)

    return chemin

```

```

In [72]: #Q17 : 5 pts
def weightWay(file_name) :
    chemin = setWay(file_name)
    return chemin.sum() - chemin.data

```

Département : Mathématiques et Informatique	Examen :	N°1
Année Universitaire : 2017/2018	D. S :	
Matière : Informatique (2 ^{ème} Année MP, PC, PT)	Date : 03/01/2018	
Nombre des pages : 4	Durée : 2h	

EXERCICE 1

En algèbre linéaire, une matrice diagonale est une matrice carrée dont les coefficients en dehors de la diagonale principale sont nuls.

Les coefficients de la diagonale peuvent être ou ne pas être nuls.

Une matrice diagonale est une matrice qui correspond à la représentation d'un endomorphisme diagonalisable dans une base de vecteurs propres.

Toute matrice diagonale est symétrique normale et triangulaire.

La matrice identité I_n carrée d'ordre n est diagonale.

• Définition

Une matrice carrée $D = (d_{i,j})_{1 \leq i, j \leq n}$ est dite diagonale si :

$$\forall (i, j) \in \llbracket 1, n \rrbracket^2, i \neq j \Rightarrow d_{i,j} = 0.$$

Exemples :

Les matrices suivantes sont diagonales :

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -i \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -3 \end{pmatrix}$$

En revanche, les matrices suivantes ne sont pas diagonales :

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & i & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & -3 \end{pmatrix}$$

• Notation

Comme une matrice diagonale est entièrement déterminée par la liste de ses éléments diagonaux, la notation suivante plus concise et souvent adoptée :

$$\text{diag}((a_1, a_2, \dots, a_n)) = \begin{pmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_n \end{pmatrix}$$

- **Propriétés**

Addition :

$$\text{diag}((a_1, a_2, \dots, a_n)) + \text{diag}((b_1, b_2, \dots, b_n)) = \text{diag}((a_1 + b_1, a_2 + b_2, \dots, a_n + b_n))$$

Multiplication :

$$\text{diag}((a_1, a_2, \dots, a_n)) * \text{diag}((b_1, b_2, \dots, b_n)) = \text{diag}((a_1 * b_1, a_2 * b_2, \dots, a_n * b_n))$$

donc

$$\forall k \in \mathbb{N}^* \text{diag}((a_1, a_2, \dots, a_n))^k = \text{diag}((a_1^k, a_2^k, \dots, a_n^k))$$

Déterminant :

$$\det(\text{diag}((a_1, a_2, \dots, a_n))) = \prod_{k=1}^n a_k$$

Inversibilité :

Une matrice diagonale est inversible si et seulement si tous les a_i sont inversibles (c'est-à-dire non nuls) et dans ce cas,

$$\text{diag}((a_1, a_2, \dots, a_n))^{-1} = \text{diag}\left(\frac{1}{a_1}, \frac{1}{a_2}, \dots, \frac{1}{a_n}\right)$$

Dans la suite on propose de construire la classe **diag** qui définit et manipule les objets matrices diagonales et dont la documentation est la suivante :

" " "

Une classe qui définit les matrices diagonales

Sommaire :

```
>>> d1=diag((1,2,3))
>>> d2=diag((4,5,6))
>>> print(d1)
diag(1.0 , 2.0 , 3.0)
>>> print(d2)
diag(4.0 , 5.0 , 6.0)
>>> print(d1+d2)
diag(5.0 , 7.0 , 9.0)
>>> print(d1*d2)
diag(4.0 , 10.0 , 18.0)
>>> print(d2**3)
diag(64.0 , 125.0 , 216.0)
>>> d1.det()
6
>>> d=diag((1,-1,0,1))
>>> d.estInversible()
False
>>> print(d2.inv())
diag(0.25 , 0.2 , 0.1666666666667)
" " "
```

Travail demandé :

Question 1

En se basant sur la documentation construire la class **diag** dans laquelle les méthodes suivantes sont implémentées (en faisant les exceptions nécessaires) :

`__str__` (ou `__repr__`), `__add__`, `__mul__`, `__pow__`, `det`, `estInversible` et `inv`

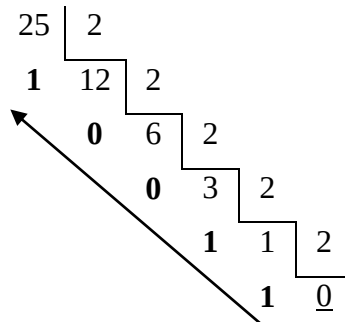
Question 2

Construire la classe **identity** qui, à partir d'un entier **n** strictement positif, définit les matrices identités d'ordre *n* et qui hérite de la classe **diag**.

EXERCICE 2

- On demande de construire la classe **stack** qui définit la structure pile (à partir d'une liste) et qui contient les méthodes suivantes :
 - `__len__` : qui retourne la taille d'une pile.
 - `isEmpty` : qui retourne `True` ou `False` suivant que la pile est vide ou non.
 - `push` : qui insère un élément au sommet de la pile.
 - `top` : qui retourne l'élément au sommet de la pile.
 - `pop` : qui supprime l'élément au sommet de la pile (si la pile est vide) et retourne sa valeur.
- On se propose d'écrire des fonctions assurant la conversion de décimaux en binaire et réciproquement en utilisant une pile.
 - Sachant que la division entière d'un décimal *d* par 2 donne le dernier chiffre en binaire, écrire la fonction python **d2b** qui retourne la pile des chiffres binaires d'un décimal *d*.

Exemple :



$$(25)_{10} = (1 \ 1 \ 0 \ 0 \ 1)_2$$

- Ecrire une fonction python **b2d** qui prend en argument une pile **p** et retourne le décimal dont **p** stockant sa représentation binaire.

Exemple :

$$(1 \ 1 \ 0 \ 0 \ 1)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 8 + 1 = (25)_{10}$$

EXERCICE 3

Soit le schéma relationnel suivant :

- Film (**Titre**, **Realisateur**, **Duree**, **Type**, **Annee**)

- Titre: titre du film (chaîne de caractères), clé primaire.
- Realisateur : nom du réalisateur (chaîne de caractères).
- Duree : durée du film (entier).
- Type : type du film (chaîne de caractères).
- Annee : année de sortie du film (entier).
- Salle (**NumS**, **Horaire**, **Capacité**, **#Titre**)
 - NumS: désignation de la salle (chaîne de caractères), clé primaire.
 - Horaire: heure du démarrage du film (chaîne de caractères).
 - Capacité: capacité de la salle (entier).
 - Titre: titre du film (chaîne de caractères), clé étrangère.
- Producteur (**Producteur**, **#Titre**)
 - Producteur : nom du producteur (chaîne de caractères), clé primaire.
 - Titre: titre du film (chaîne de caractères), clé étrangère.
- Vu (**IdSpectateur**, **#Titre**)
 - IdSpectateur : nom du Spectateur (chaîne de caractères), clé primaire.
 - Titre: titre du film (chaîne de caractères), clé étrangère.

Travail demandé :

Ecrire les requêtes SQL permettant de donner :

Question 1

Créer les tables Film et Vu en respectant toutes les contraintes mentionnées ci-dessus.

Question 2

Ajouter dans la table film le film 'la révolution', réalisé en 2011 par le 'le révolutionnaire', et qui est un 'long métrage' de durée 1 heure.

Question 3

Les noms des films produits en 2016.

Question 4

Où et à quel horaire peut-on voir le film 'la révolution'.

Question 5

Le(s) nom(s) de la (des) salle(s) ayant la capacité maximale.

Question 6

La moyenne des durées des films.

Question 7

Le nombre de films produits par chaque producteur.

Question 8

Le nombre de spectateurs par film.

Question 9

Le nombre de spectateurs du film 'la révolution'.

Alternative de correction de l'Examen n°1

EXERCICE 1 (10 points = 8 points + 2 points)

Question 1 (8 points = 1 point par méthode)

```
>>> class diag:
```

```
    def __init__(self,t):
```

```
        self.data=t
```

```
    def __str__(self):
```

```
        ch="diag({}".format(float(self.data[0]))
```

```
        for i in range(1,len(self.data)):
```

```
            ch+=", "+str(float(self.data[i]))
```

```
        ch+=")"
```

```
        return ch
```

```
    def __add__(self,other):
```

```
        try:
```

```
            assert len(self.data)==len(other.data)
```

```
            t=()
```

```
            for i in range(len(self.data)):
```

```
                t+=(self.data[i]+other.data[i],)
```

```
            return diag(t)
```

```
        except:
```

```
            print("Erreur, dimensions incompatibles")
```

```
    def __mul__(self,other):
```

```
        try:
```

```
            assert len(self.data)==len(other.data)
```

```
            t=()
```

```
            for i in range(len(self.data)):
```

```

        t+=(self.data[i]*other.data[i],)

    return diag(t)

except:

    print("Erreur, dimensions incompatibles")

def __pow__(self,p):

    try:

        assert type(p)==int and p>1

        t=()

        for i in range(len(self.data)):

            t+=(self.data[i]**p,)

        return diag(t)

    except:

        print("Erreur, p non entier")

def det(self):

    p=1

    for e in self.data:

        p*=e

    return p

def estInversible(self):

    return self.det()!=0

def inv(self):

    try:

        assert self.estInversible()==True

        t=()

        for i in range(len(self.data)):

            t+=(1/float(self.data[i]),)

        return diag(t)

```

```
except:
```

```
    print("Erreur, la matrice est non inversible")
```

Question 2 (2 points)

```
>>> class identity(diag):
```

```
    def __init__(self,n):
```

```
        L=[1 for i in range(n)]
```

```
        diag.__init__(self,tuple(L))
```

EXERCICE 2 (5 points = 2 points + 1,5 points + 1,5 points)

1)

```
>>> class stack:
```

```
    def __init__(self,L=[]):
```

```
        self.elets=L
```

```
    def __len__(self):
```

```
        return len(self.elets)
```

```
    def isEmpty(self):
```

```
        return self.elets==[]
```

```
    def push(self,e):
```

```
        self.elets.append(e)
```

```
    def top(self):
```

```
        return self.elets[-1]
```

```
    def pop(self):
```

```
        if not(self.isEmpty()):
```

```
            e=self.elets.pop()
```

```
            return e
```

```
        else:
```

```
            print("pile vide")
```

2)

```
>>> def d2b(d):  
    p=stack()  
    q=d  
    while q!=0:  
        r=q%2  
        p.push(r)  
        q=q//2  
    return p  
  
>>> def b2d(p):  
    poids=len(p)  
    d=0  
    while not(p.isEmpty()):  
        e=p.pop()  
        d+=e*2**(poids-1)  
        poids-=1  
    return d
```

EXERCICE 3 (5 points = 0,5 point par requête)

Question 1

```
CREATE TABLE Film(Titre TEXT PRIMARY KEY,Realisateur  
                    TEXT,Duree INTEGER,Type TEXT,Annee INTEGER) ;  
  
CREATE TABLE Vu(IdSpectateur INTEGER PRIMARY KEY,Titre  
                  TEXT FOREIGN KEY REFERENCES Film);
```

Question 2

```
INSERT INTO TABLE Film  
  
VALUES('la révolution',le révolutionnaire',1,'long  
      métrage',2011);
```

Question 3

```
SELECT  Titre  
  
FROM Film ;  
  
WHERE Annee=2016 ;
```

Question 4

```
SELECT  NumS , Horaire  
  
FROM Salle ;  
  
WHERE Titre='la révolution' ;
```

Question 5

```
SELECT  NumS  
  
FROM Salle ;  
  
WHERE Capacite=SELECT MAX(Capacite) FROM Salle ;
```

Question 6

```
SELECT AVG(Duree)  
  
FROM Film
```

Question 7

```
SELECT Producteur , COUNT(*)  
  
FROM Producteur  
  
GROUP BY Producteur ;
```

Question 8

```
SELECT Titre , COUNT(*)  
  
FROM Vu  
  
GROUP BY Titre ;
```

Question 8

```
SELECT COUNT(*)  
  
FROM Vu , Film  
  
GROUP BY Titre = 'la révolution' ;
```



Département : Mathématiques et Informatique	Examen :	N°1
Année Universitaire : 2018/2019	D. S :	
Matière : Informatique (2 ^{ème} Année MP, PC, PT)	Date : 12/12/2018	
Nombre des pages : 3	Durée : 2h	

PROBLEME 1

Dans une entreprise plusieurs personnes doivent avoir accès à une imprimante partagée sur un réseau local. Cette imprimante porte un nom et caractérisée par le nombre de pages pouvant l'imprimer à une minute. Les employés génèrent des tâches d'impression en général comprises entre 1 et 20 pages. Le nombre de tâches peut être estimé à environ une tâche toutes les deux minutes.

Partie 1

Une file d'attente est une structure de données abstraite et linéaire basée sur le principe du « premier arrivé, premier sorti » ou FIFO (First In, First Out). Ainsi dans une file d'attente, la première personne entrée dans la file sera la première personne à être servie.

1. Construire la classe **file** à partir d'une liste (attribut) en implémentant les méthodes de classe suivantes :
 - **__init__(...)** : qui permet de créer une file vide correctement initialisée ;
 - **empty(...)** : qui retourne *True* si une file est vide et *False* sinon ;
 - **enfiler(...)** : qui permet de placer un objet à l'arrière de la file ;
 - **defiler(...)** : qui permet d'enlever l'élément se trouvant en tête de la file et le retourner si la file est vide.

Partie 2

On propose de construire les classes **printer** et **task** ».

Description des classes

- La classe **task** est caractérisée par :
 - *heure de création* (une valeur entière)
 - *nombre de pages* à imprimer (une valeur aléatoire entre 1 et 20), on propose l'utilisation de la fonction **randrange** de la bibliothèque **random** pour initialiser cet attribut).

Méthode :

- **taskWaitingTime(...)** : qui calcule et retourne le temps d'attente d'une tâche avant son exécution par rapport au temps courant «timeNow » passé en paramètres.

Exemple :

T = task(5) est une instance de la classe **task** créée à l'instant 5.

T.taskWaitingTime(8) retourne 3.

- La classe **printer** a comme propriétés :
 - la *vitesse d'impression* **printSpeed** et le *nom de l'imprimante* qui doivent être fournis au constructeur.
 - de plus et de manière interne, un tel objet gère la *tâche d'impression courante* **currentTask** et le *temps restant* **remainingTime** pour accomplir ce travail.

Le *temps restant* est initialisé à 0 tandis que la *tâche courante* est initialisée à None.

En effet, la propriété *tâche courante* **currentTask** contiendra comme valeur un objet de type **task** et au moment de la création de l'imprimante, celle-ci ne possède pas encore de tâche d'impression.

Méthodes :

- **busy(...)** : qui vérifie si l'imprimante est occupée ou non. Pour tester si l'imprimante est occupée, il suffit de consulter l'état de la propriété **currentTask**.
- **tick(...)** : qui simule l'avancement du temps d'une seconde (incrément de 1). À chaque **tick()**, le temps restant nécessaire à l'impression de la tâche en cours est diminué d'une seconde. Si ce temps devient 0, la tâche d'impression est supprimée.
- **startNextJob(...)** : qui simule l'impression d'une tâche passée en paramètres. Lorsque l'imprimante démarre un travail d'impression, cette tâche est mémorisée dans **currentTask** et le temps nécessaire à l'impression est calculé en secondes en divisant le nombre de pages à imprimer par sa caractéristique physique **printSpeed**.

Exemple :

`P = printer('Bureau', 10)` est une instance de la classe **printer** dont le nom est 'Bureau' et la vitesse est 10.

Travail demandé

1. construire la classe **task**.
2. construire la classe **printer**.

Partie 3

La simulation d'impression fonctionne en simulant l'écoulement du temps seconde par seconde (le temps est initialement 0 et s'incrémente de 1). Au départ, l'imprimante est créée et la file d'attente est vide. À chaque avancement du temps (incrément de 1), on :

- ✓ vérifie si une nouvelle tâche d'impression doit être créée.
Comme une tâche est censée être générée toutes les deux minutes, elle est créée si le nombre entier aléatoire compris entre 1 et 120 vaut un. Dans ce cas, cette tâche est placée dans la file d'attente associée à l'imprimante ;
- ✓ teste si l'imprimante n'est pas occupée et si sa file d'attente n'est pas vide. Dans ce cas, la tâche en tête de la file sert à créer un nouveau job d'impression qui est démarré et cette tâche est enlevée de la file d'attente.

Le temps que cette tâche a passé dans la file d'attente est placé dans la liste des temps d'attente. Cette liste sert à la fin pour calculer le temps moyen d'attente des tâches pour être imprimées. Dans tous les cas, le temps de l'imprimante est avancé d'une seconde entraînant éventuellement la terminaison du job d'impression en cours.

Travail demandé

1. Ecrire la fonction python nommée, **newPrintTask** qui permet de vérifier si une nouvelle tâche d'impression doit être créée (si un nombre entier aléatoire compris entre 1 et 120 vaut un).

2. Ecrire la fonction python nommée, **simulation** qui permet de :
 - Créer une imprimante avec le nom '**Bureau**' et la vitesse **ppm** passée en paramètres.
 - Créer une file de tâches vide et la liste **waitingTime** initialement vide.
 - Simuler l'impression en suivant la démarche décrite ci-dessus pour une durée **nbrSeconds** passée en paramètres.
 - Calculer et afficher le temps moyen d'attente des tâches.
3. Ecrire les instructions python permettant d'effectuer deux simulations de 2 heures (120 secondes) pour les valeurs de **printSpeed** 6 et 20.

PROBLEME 2

Une société multinationale, commercialisant des produits chimiques, possède une base de données pour le stockage de ses produits, dont le schéma relationnel est le suivant :

- **Hangar** (id_hangar, nom, ville, pays)
- **Ligne** (lig, id_hangar, capacité)
- **Categorie** (id_categorie, libelle, dangerosite)
- **Produit** (id_produit, nomProduit, id_categorie)
- **Conteneur** (id_conteneur, volume, lig, id_produitl)

Avec:

- Un hangar est structuré en lignes
- Une ligne possède une capacité, exprimée en litres représentant le volume maximal qui peut y être stocké
- Un conteneur est rangé dans un hangar quand conteneur.lig est non NULL.
- un conteneur est vide quand conteneur.id_produit vaut NULL
- Pour des raisons de sécurité, les conteneurs rangés dans la même ligne stockent obligatoirement des produits de la même catégorie.

Partie 1

Ecrire les requêtes SQL donnant :

1. La liste des hangars de 'Tunis' ;
2. Les hangars qui possèdent le volume maximal stocké des produits de la catégorie 'liquides inflammables' ;
3. les hangars avec le volume stocké et le nombre de conteneurs qu'ils contiennent (0 si hangar vide) ;
4. Les hangars, de 'Algérie', qui ont de la place pour stocker le conteneur dont l'identifiant est égale à 201820 ;
5. Le nom de la catégorie la plus dangereuse stockée dans des hangars dont le nom commence par la lettre 'Tu' et l'avant dernier caractère est '3' ;
6. les hangars qui stockent les produits les plus dangereux ;
7. Pour chaque pays quel est le produit qui est le plus disponible ;

8. Le nombre de conteneurs vides par ville ;
9. la liste des hangars selon l'ordre croissant du volume (en litre) occupé ;
10. La moyenne de l'occupation en volume par hangar (en ordre croissant) ;
11. La liste des lignes non entièrement occupés par catégorie de produit pour le hangar du nom 'Tunis1234' ;
12. les hangars qui stockent les mêmes produits que ceux stockés à 'Médénine'.

Partie 2

Répondre en algèbre relationnelle aux questions : 1, 4 et 12 de la **Partie 1**.

Corrigé Examen 1

PROBLEME 1

Partie 1

1. Construction de la classe file

```
class file:
    def __init__(self):
        self.items = []
    def enfiler(self, item):
        self.items.append(item)
    def empty(self):
        return self.items == []
    def defiler(self):
        return self.items.pop(0)
```

Partie 2

1. Construction de la classe task

```
import random
class task:
    def __init__(self, temps):
        self.heureCreation = temps
        self.nbPages = random.randrange(1,21)
    def taskWaitingTime(self, tempsCourant):
        return tempsCourant - self.heureCreation
```

2. Construction de la classe printer

```
class printer:
    def __init__(self,nom,ppm):
        self.printerSpeed = ppm
        self.currentTask = None
        self.remainingTime = 0
        self.nomImprimante = nom
    def busy(self):
        if self.currentTask != None:
            return True
        else:
            return False
    def tick(self):
        if self.currentTask != None:
            self.remainingTime = self.temps_restant - 1
            if self.remainingTime <= 0:
                self.currentTask = None
    def startNextJob(self,tache):
        self.currentTask = tache
        self.remainingTime = (tache.nbPages * 60) /
                               self.printerSpeed
```

Partie 3

1.

```
def newPrinterTask():
    num = random.randrange(1,120)
    if num == 1:
        return True
    else:
        return False
```

2.

```
def simulation(nbSecondes,vitesse):
    p = printer('Bureau',vitesse)
    fileTaches = file()
    listAttente = []
    for seconde in range(nbSecondes):
```

```
        if newPrinterTask():
            t = task(seconde)
            fileTaches.enfiler(t)
            if (not p.busy()) and (not fileTaches.empty()):
                t=fileTaches.defiler()
                listAttente.append(t.remainingTime
(seconde))
            p.startNextJob(t)
            p.tick()
            return sum(listAttente)/len(listAttente)
```

3.

```
print(simulation(7200,6))
print(simulation(7200,20))
```

PROBLEME 2

Partie 1

1)

```
SELECT *
FROM Hangar
WHERE ville ='Tunis'
```

2)

```
SELECT *
FROM Hangar H , Conteneur C , Ligne L
WHERE H.id_hangar = L.id_hangar
AND L.lig=C.lig
AND pays="Algérie"
```

3)

```
SELECT P.nomProduit
FROM Produit P , Hangar H , Conteneur C , Ligne L
WHERE H.id_hangar = L.id_hangar
AND L.lig=C.lig
AND P.id_produit=C.id_produit
AND H.nom like 'Tu%3_'
```

- 4)
- ```
SELECT *
FROM Ligne
ORDER BY id_hangar DESC , capacite ASC
```
- 5)
- ```
SELECT *
FROM Produit P , Hangar H , Conteneur C , Ligne L , Categorie Ca
WEHER H.id_hangar=L.id_hangar
AND L.lig=C.lig
AND P.id_produit=C.id_produit
AND Ca.id_categorie=P.id_categorie
AND Ca.libelle="liquides inflammables"
```
- 6)
- ```
SELECT *
FROM Produit P , Categorie C
WHERE P.id_categorie=C.id_categorie
AND C.dangerosite = (SELECT MAX(dangerosite) FROM Categorie)
```
- 7)
- ```
SELECT H.ville , COUNT(*)
FROM Hangar H , Ligne L , Conteneur C
WHERE H.id_hangar = L.id_hangar
AND L.lig=C.lig
AND C.id_produit is not null
GROUP BY H.ville
```
- 8)
- ```
SELECT L.id_hangar , SUM(C.volume)
FROM Ligne L , Conteneur C
WHERE L.lig = C.lig
GROUP BY L.id_hangar
```
- 9)
- ```
SELECT L.id_hangar , AVG(C.volume) moy
FROM Ligne L , Conteneur C
WHERE L.lig = C.lig
GROUP BY L.id_hangar
ORDER BY moy;
```

10)

```

SELECT sum(C.volume) s , pays , P.NomProduit
FROM Ligne L , Conteneur C , Hangar H , Produit P
WHERE L.lig = C.lig
AND H.id_hangar=L.id_hangar
AND C.id_produit = P.id_produit
GROUP BY H.pays , C.id_produit
HAVING s >=
(SELECT SUM(C.volume)
FROM Ligne L , Conteneur C , Hangar H
WHERE L.lig=C.lig
AND H.id_hangar = L.id_hangar
GROUP BY H.pays , id_produit)

```

11)

```

SELECT *
FROM Hangar H , Ligne L , Conteneur C
WHERE H.id_hangar = L.id_hangar
AND L.lig = C.lig
AND C.id_produit in
(SELECT C.id_produit
FROM Hangar H , Ligne L , Conteneur C
WHERE H.id_hangar = L.id_hangar
AND L.lig = C.lig
AND H.ville = "Mednine")

```

Partie 2

1)

 $\sigma_{ville='Tunis'}(Hangar)$

2)

 $(\sigma_{pays='Algérie'}(Hangar)) \bowtie_{id_hangar} (Ligne) \bowtie_{lig} (Conteneur)$

3)

 $R1 = \pi_{id_produit} ((\sigma_{H.ville='Mednine'}(Hangar)) \bowtie_{id_hangar} (Ligne) \bowtie_{lig} (Conteneur))$
 $R2 = (\sigma_{H.ville \neq 'Mednine'}(Hangar)) \bowtie_{id_hangar} (Ligne) \bowtie_{lig} (Conteneur)$
 $R1 \bowtie_{id_produit} R2$

Département : Mathématiques et Informatique	Examen :	N°2
Année Universitaire : 2018/2019	D. S :	
Matière : Informatique (2 ^{ème} Année MP, PC, PT)	Date : 30/04/2019	
Nombre des pages : 4	Durée : 2h	

PROBLEME 1

Dans ce problème on s'intéresse à l'implémentation d'un algorithme de détection de contours appliqué sur les images en niveau de gris. On rappelle qu'une image en niveau de gris est représentée par une matrice où les coefficients sont les valeurs des pixels.

La détection des contours est un des problèmes les plus fondamentaux et les plus importants dans le domaine de traitement des images numériques. Les contours sont des pixels de l'image caractérisés par un fort contraste (un saut substantiel en intensité lumineuse). Un algorithme de détection de contours produit en sortie une image binaire tel que appelée carte des contours où les points blancs marquent les frontières entre les régions de l'image. Ainsi, la détection des contours permet de réduire les données de l'image tout en préservant les propriétés structurales de cette dernière.

Dans ce qui suit, on suppose avoir déjà effectué les importations suivantes :

```
import numpy as np
import matplotlib.pyplot as plt
```

1. Ecrire une fonction python nommée **load_image** qui à partir d'un nom physique d'une image en niveau de gris représenté par **FImg** retourne la matrice de pixels associée.

L'opération de filtrage linéaire correspond à une convolution de la matrice de l'image source **S** par un masque **M** (une matrice carrée d'ordre n), cette opération produit une matrice **R** de même taille que **S**. Dans ce problème, on considère uniquement les masques carrés tel que n est impair. La convolution consiste à coulisser la matrice **M** sur la matrice **S**. L'application du masque **M** sur **S** produit une autre image **R** tel que :

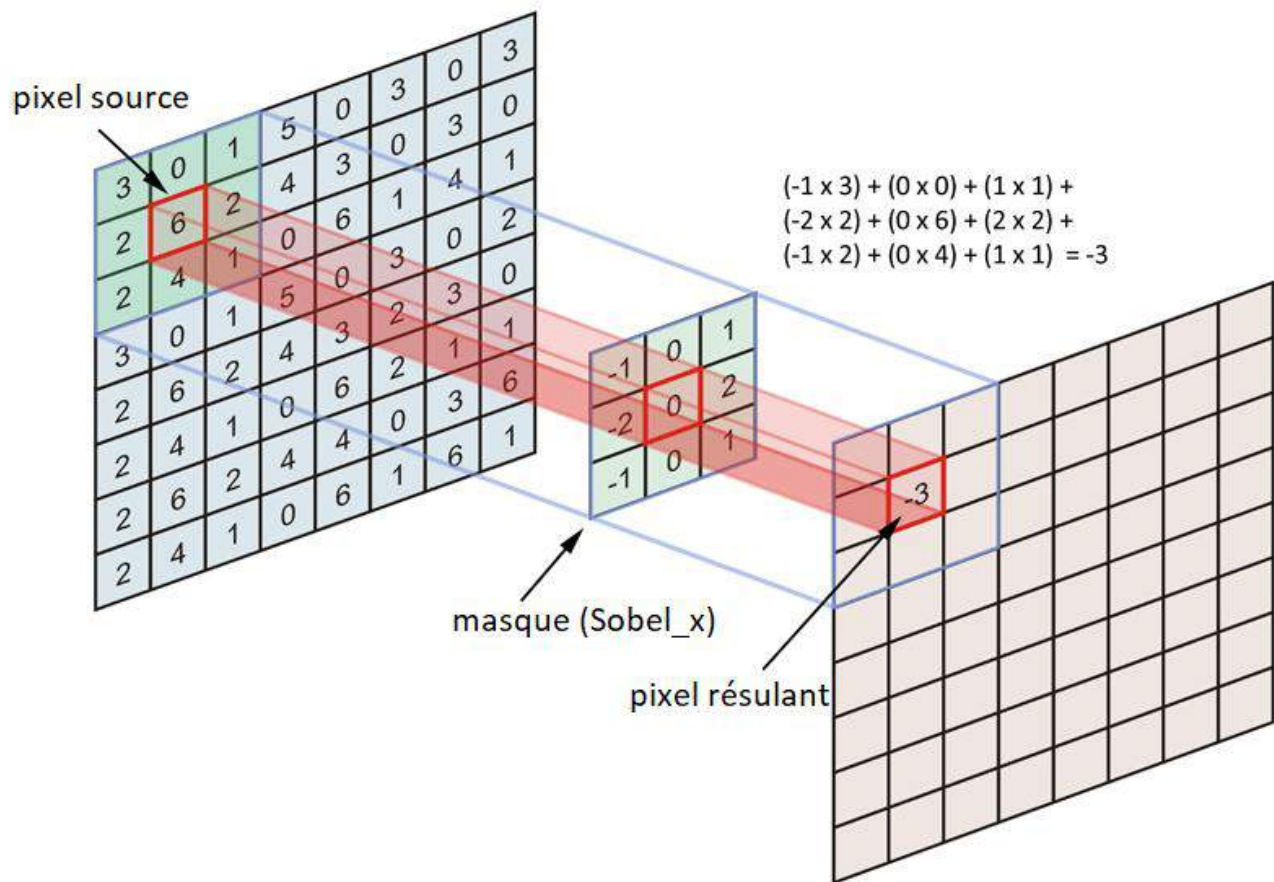
$$R_{i,j} = \sum_{x=-n\div 2}^{n\div 2} \sum_{y=-n\div 2}^{n\div 2} M_{x,y} \times S_{i+x,j+y} \quad (1)$$

Noter que les indices d'un masque représentent le décalage par rapport au point central du masque.

Exemple : masque de taille 3×3

$M_{-1,-1}$	$M_{-1,0}$	$M_{-1,1}$
$M_{0,-1}$	$M_{0,0}$	$M_{0,1}$
$M_{1,-1}$	$M_{1,0}$	$M_{1,1}$

Exemple : illustration d'un produit de convolution avec un masque de taille 3×3



Un problème se manifeste aux bords de l'image **S**, pour résoudre ce problème la formule décrite dans (1) est remplacée par :

$$R_{i,j} = \sum_{x=-ndiv2}^{ndiv2} \sum_{y=-ndiv2}^{ndiv2} M_{x,y} \times S_{k,l} \text{ où } \begin{cases} k = (i+x) \bmod n \\ l = (j+y) \bmod m \end{cases} \text{ avec } (n,m) \text{ les dimensions } S$$

Dans la suite on propose de construire la classe **Masque** permettant de représenter la matrice d'un filtre linéaire en respectant la description suivante :

2. Implémenter le constructeur permettant d'initialiser un attribut nommé **data** qui est une matrice supposée carrée d'ordre **n** impair.
3. Définir la méthode **__getitem__** qui à partir d'un tuple formé par un couple d'entiers (**i** , **j**) retourne la valeur de l'élément décalé de **i** lignes et de **j** colonnes par rapport au point central du masque.

Exemple :

```
M = Masque(np.reshape(np.arange(9), (3,3)))
```

```
M[0,0] donne 4
```

```
M[-1,-1] donne 0
```

```
M[1,1] donne 8
```

4. Définir la méthode **__setitem__** qui à partir d'un tuple formé par un couple d'entiers (**i** , **j**) et d'un réel **r** remplace la valeur de l'élément d'indice **i** et **j** du masque par **r**.

5. Définir la méthode **indices** qui retourne une liste de tuples contenant tous les indices possible d'un masque.

Exemple :

M.indices() donne $[(-1,1),(-1,0),(-1,-1),(0,-1),(0,0),(0,1),(1,-1),(1,0),(1,1)]$

6. Définir la méthode **__mul__** qui permet de retourner la matrice résultat de la convolution d'un masque par une matrice donnée en utilisant la méthode décrite ci-dessus.

Application :

7. Ecrire une fonction gaussien qui à partir d'un entier n (supposé impair) et d'un réel sigma retourne un objet instance de la classe Masque contenant un filtre gaussien de taille n et de variance sigma tel que ces éléments sont calculés par la fonction suivante :

$$G_{\sigma}(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

où (x,y) désignent le décalage par rapport à l'élément central du masque.

PROBLEME 2

Le schéma relationnel suivant permet de stocker des informations relatives à la gestion d'une base de données.

- **Utilisateurs** (IdU , Nom , Prenom)
La relation *Utilisateur* contient tous les utilisateurs de la base.
 - IdU : identifiant de l'utilisateur (entier), clé primaire.
 - Nom : nom de l'utilisateur (chaîne de caractères).
 - Prenom : prénom de l'utilisateur (chaîne de caractères).
- **Table** (IdTable , IdCreateur)
La relation *Table* contient toutes les tables de la base.
 - IdTable : nom de la table (chaîne de caractères), clé primaire.
 - IdCreateur : identifiant du créateur de la table (entier).
- **Privilege** (IdTable , IdU , Droit)
La relation *privilege* définit les droits de manipulation des tables identifiées par IdTable par les utilisateurs identifiés par IdU.
 - IdTable , IdU: clé primaire.
 - Droit appartient à { 'CREATE', 'DROP', 'ALTER', 'SELECT', 'INSERT', 'UPDATE', 'DELETE', 'ALL', ... }.

Travail demandé :

Question 1

En utilisant le module sqlite3 donner le script python permettant de créer la table **Utilisateurs** dans la base "GestionBDD.db" en exprimant toutes les contraintes d'intégrité mentionnées ci-dessus.

Question 2

Ecrire les instructions python permettant de saisir les informations relatives à un utilisateur quelconque puis les insérer dans la table **Utilisateurs**.

Dans la suite on suppose que les trois tables de la base sont créées et remplies. Donner les requêtes SQL permettant de :

Question 3

Déterminer le nom et le prénom de tous les utilisateurs, Classés par ordre alphabétique décroissant selon les noms.

Question 4

Donner le nombre des utilisateurs de la base.

Question 5

Déterminer, pour chaque créateur, le nombre de tables créées.

Question 6

Déterminer les identifiants des utilisateurs ayant le droit de création de nouvelles tables.

Question 7

Déterminer les noms et les prénoms des utilisateurs ayant le droit 'INSERT' associé à la table dont IdTable = 'Produit'.

Ecrire en algèbre relationnelle les expressions suivantes :

Question 8

Déterminer les identifiants des créateurs de la table dont IdTable = 'Films'.

Question 9

Déterminer les noms et les prénoms des utilisateurs ayant tous les droits d'accès à la table dont IdTable = 'Salles'.

Question 10

Déterminer les identifiants des utilisateurs qui sont aussi des créateurs.

Corrigé Examen 2

PROBLEME 1

```
import numpy as np
import Matplotlib.pyplot as plt
def load_image(FImg):
    return plt.imread(FImg)
```

Construction de la classe Masque

```
class Masque:
    def __init__(self, M):
        self.data = M
    def __getitem__(self, (i,j)):
        n = np.shape(self.data)[0]
        return self.data[i+(n//2),j+(n//2)]
    def indices(self):
        n = np.shape(self.data)[0]
        L = []
        for i in range(-n//2,n//2+1):
            for j in range(-n//2,n//2+1):
                L.append((i,j),)
        return L
    def __mul__(self,S):
        (l,c) = np.shape(S)
        R = np.empty((l,c))
        for i in range(l):
            for j in range(c):
                R[i,j]=sum(self[x,y]*S[(i+x)%l, (j+y)%c]
                            for y in range(-n//2,n//2+1)
                            for x in range(-n//2,n//2+1))
        return R
```

Application

```
def gaussian(n , sigma):  
    M = Masque(np.zeros((n,n))  
    L = M.indices()  
    for (i,j) in L :  
        M[i,j] = 1/(2*np.pi*sigma**2)*np.exp(-  
            (i**2+j**2)/(2*sigma**2))  
  
    return M
```

PROBLEME 2**Question 1**

```
import sqlite3  
  
conn = sqlite3.connect('GestionBDD.db')  
cur = conn.cursor()  
Req = "Create Table Utlistaeurs(IdU integer primary key ,  
Nom Text , Prenom Text)"  
cur.execute(Req)
```

Question 2

```
IdU = int(input("saisir l'identifiant : "))  
Nom = input("saisir le nom : ")  
Prenom = input("saisir le prénom : ")  
Req = "Insert into Utlistaeurs  
Values({}, {}, {})".format(IdU, Nom, Prenom)  
cur.execute(Req)
```

Question 3

```
SELECT Nom , Prenom  
FROM Utlistaeurs  
ORDER BY Nom DESC
```

Question 4

```
SELECT Count(*)  
FROM Utlistaeurs
```

Question 5

```
SELECT IdCreateur , Count(*)  
FROM Table  
GROUP BY IdCreateur
```

Question 6

```
SELECT IdU  
FROM Privilege  
WHERE Droit in ("Create" , "All")
```

Question 7

```
SELECT Nom , Prenom  
FROM Privilege , Utilisateurs  
WHERE Privilege.IdU = Utilisateurs.IdU AND IdTable =  
"Produit" AND Droit in ("Insert" , "All")
```

Question 8
$$\Pi(\mathbf{IdCreateur})(\sigma_{\mathbf{IdTable}=\mathbf{"Films"}}(\mathbf{Table}))$$
Question 9
$$\Pi(\mathbf{Nom},\mathbf{Prenom})(\sigma_{\mathbf{IdTable}=\mathbf{"Salles"}} \text{ et } \mathbf{droit}=\mathbf{"All"}}(\mathbf{Privilege} \bowtie_{\mathbf{IdU}} \mathbf{Utilisateurs}))$$
Question10
$$\Pi(\mathbf{IdU})(\mathbf{Utilisateurs}) \cap \Pi(\mathbf{IdCreateur})(\mathbf{Table})$$

Département : Mathématiques et Informatique	Examen :	N°3
Année Universitaire : 2016/2017	D. S :	
Matière : Informatique (2 ^{ème} Année MP, PC, PT)	Date : 25/04/2017	
Nombre des pages : 3	Durée : 2h	

EXERCICE 1 (9 points)

A.

Une pile est une structure de données composée de type LIFO (Last In First Out) et qui permet uniquement les primitives suivantes :

- **taille** : retourne la taille de la pile.
- **estVide** : retourne vrai si la pile est vide et faux sinon.
- **sommet** : retourne l'élément sommet de la pile (le dernier entré).
- **empiler** : ajoute au sommet de la pile un élément.
- **depiler** : supprime de la pile le sommet.

Question 1

On propose de construire à partir d'une liste la classe Pile définissant toutes les primitives mentionnées ci-dessus comme étant méthodes de classe.

Le constructeur de la classe se présente comme suit :

```
def __init__(self, L = []):  
    self.stack=L
```

B.

Une maîtresse d'école décide d'instaurer un système de points à la participation des élèves à un examen. A chaque élève, est associée une pile qui mémorise les différentes réponses. Si la réponse est correcte, alors un feu vert est empilé sur sa pile, si la réponse est approximative alors un feu orange est empilé et si la réponse est incorrecte alors un feu rouge est empilé.

Chaque élève a 0 point initialement, puis les points sont comptés de la façon suivante : si la réponse est correcte, alors le nombre de points de l'élève est incrémenté de 10 points, si la réponse est approximative, alors le nombre de points est incrémenté de 5 points et si la réponse est incorrecte alors le nombre de points est décrémenté de 10 points.

On suppose qu'on a n élèves numérotés de 1 à n , les réponses sont codées par des lettres avec 'V' représente le feu vert, 'O' représente le feu orange et 'R' représente le feu rouge et que les élèves ainsi que leurs piles de réponses sont stockés dans un dictionnaire d tels que les clés sont les numéros des élèves et les valeurs sont les piles de réponses.

Exemple :

$$d = \{1: ['V', 'V', 'O', 'R', \dots], 2: ['R', 'V', 'O', 'R', \dots], \dots, n: ['O', 'V', 'O', 'R', \dots]\}$$

Question 2

Ecrire une fonction PYTHON nommée **nombre** qui saisit et retourne le nombre des élèves enseignés par la maîtresse.

Question 3

Ecrire une fonction PYTHON nommée **saisiRep** qui pour chaque élève saisit sa pile de réponses et retourne le dictionnaire **d**.

Question 4

Ecrire une fonction **afficheRep** qui à partir du dictionnaire **d** affiche pour chaque élève ses réponses ('correcte', 'approximative' ou 'incorrecte').

Question 5

Ecrire une fonction PYTHON nommée **resultat** qui à partir d'une pile de réponses **p** calcule et retourne la somme de points cumulée.

Question 6

Ecrire une fonction PYTHON nommée **resultats** qui à partir du dictionnaire **d** retourne le dictionnaire **d1** stockant pour chaque élève la somme de points cumulée suivant sa pile de réponse.

Exemple :

$$d1 = \{1: 30, 2: 75, \dots, n: 5\}$$

Aide : len(L), append(e), pop(), update({clé : valeur}), keys(), values(), items()

EXERCICE 2 (6 points)

Soient M une matrice carrée d'ordre n et $tr(M)$ sa trace ($tr(M) = \sum_{i=1}^n M_{i,i}$).

On souhaite simuler le calcul des coefficients du polynôme caractéristique de M sans utiliser les fonctions prédéfinies du calcul matriciel.

Le polynôme caractéristique P de M , est donné par :

$P = x^n + C_1 x^{n-1} + C_2 x^{n-2} + \dots + C_{n-1} x + C_n$ où les C_i sont les coefficients de P selon les puissances décroissantes en x .

Pour cela, on calcule la suite des matrices A_i ainsi que leurs traces $tr(A_i)$ définies par :

$$\begin{cases} A_1 = M, & C_1 = -tr(A_1) \\ A_i = (A_{i-1} + C_{i-1} Id) M, & C_i = \frac{-tr(A_i)}{i} \text{ pour } 2 \leq i \leq n \end{cases}$$

Id étant la matrice identité d'ordre n .

Travail demandé

Question 1

Ecrire une fonction PYTHON nommée **trace** qui retourne la trace d'une matrice **M** carrée d'ordre **n**.

Question 2

Ecrire une fonction PYTHON nommée **identite** qui crée et retourne une matrice identité d'ordre **n**.

Question 3

Ecrire une fonction PYTHON nommée **Produit** qui retourne la matrice produit de deux matrices **M** et **N** carrées d'ordre **n**.

Question 4

Ecrire une fonction PYTHON nommée **calculPol** qui retourne la liste des coefficients du polynôme caractéristique d'une matrice **M** carrée d'ordre **n**.

Aide : `shape(M)`, `empty((n,n))`, `append(e)`

EXERCICE 3 (5 points)

Soient les relations suivantes de la base de données d'une société :

Employe(NumE, NomE , Fonction , Embauche , Salaire , Comm , NumD#)
Département(NumD, NomD , Lieu)

Travail demandé

Ecrire les requêtes SQL permettant de :

- 1) Donnez tous les employés ayant une commission non NULL classés par commission décroissante.
- 2) Donnez les noms des personnes ayant un salaire supérieur à 1500 DT.
- 3) Donnez tous les employés qui travaillent à 'Créteil'.
- 4) Donnez la moyenne des salaires.
- 5) Donnez le nombre de commissions non NULL.
- 6) Donner le salaire minimum et le salaire maximum pour chaque département.
- 7) Donnez la liste des employés gagnant plus que la moyenne des salaires de l'entreprise.

Alternative de correction de l'EXAMEN 3

EXERCICE 1

A.

Question 1

class Pile :

''' une pile est une structure de composée de type LIFO '''

def __init__(self , L = []):

self.stack = L

def Taille(self):

return len(self.stack)

def EstVide(self):

return self.stack == []

def Sommet(self):

return self.stack[-1]

def Empiler(self , e):

self.stack.append(e)

def Depiler(self):

if not(self.EstVide()):

e = self.stack.pop()

return e

B.

Question 2

def Nombre():

while (True):

n = int(input(" donner le nombre d'élèves : "))

if n > 1:

break

```
return n
```

Question 3

```
def SaisiRep(n) :  
    d = {}  
    for i in range(1 , n+1) :  
        p = Pile()  
        while (True)  
            rep = input(" donner une réponse : ")  
            if rep = 'O' or rep = 'R' or rep = 'V' :  
                p.Emplier(rep)  
            elif rep = '*'  
                break  
            d.update({ i : p }) #ou bien d[i] = p  
    return d
```

Question 4

```
def AfficheRep(d) :  
    for num in d.keys() :  
        p = d[num]  
        while not(p.EstVide()) :  
            rep = p.Depiler()  
            if rep = 'V' :  
                print(" réponse correcte ")  
            elif rep = 'O' :  
                print(" réponse approximative ")  
            else :  
                print(" réponse incorrecte ")
```

Question 5

```
def resultat(p) :  
    R = 0  
    while not(p.EstVide()) :  
        rep = p.Depiler()  
        if rep == 'V' :  
            R += 10  
        elif rep == 'O' :  
            R += 5  
        else :  
            R -= 10  
    return R
```

Question 6

```
def Resultats(d) :  
    d1 = {}  
    for cle in d.keys() :  
        d1.update( {cle : resultat(d[cle])} )  
    return d1
```

EXERCICE 2

```
from numpy import *
```

Question 1

```
def Trace(M) :  
    return sum(M[i][i] for i in range(len(M)))
```

Question 2

```
def Identite(n) :  
    Id = zeros( (n,n) )  
    for i in range(n) :
```

```

        Id[i][i] = 1
    return Id

```

Question 3

```

def Produit(M , N) :
    n = len(M)
    X = empty( (n,n) )
    for i in range(n) :
        for j in range(n) :
            sum(M[i][k] * N[k][j] for k in range(n) )
    return X

```

Question 4

```

def CalculPol(M) :
    n = len(M)
    A = M[:, :]
    C = - Trace(A)
    L = [1 , C]
    for i in range(2 , n+1) :
        B = Produit((A + C*Identite(n)) , M)
        C = - Trace(B)
        L.append(C)
        A = B[:, :]
    return L

```

EXERCICE 3

- 1) SELECT * FROM Employe
WHERE Comm IS NOT NULL
ORDER BY Comm DESC ;
- 2) SELECT NomE FROM Employe
WHERE Salaire > 1500 ;
- 3) SELECT * FROM Employe , Departement
WHERE Lieu = 'Créteil ' AND Employe.NumD =
Departement.NumD ;
- 4) SELECT AVG(Salaire) Employe ;
- 5) SELECT COUNT(*) FROM Employe
WHERE Comm IS NOT NULL ;
- 6) SELECT NumD , MIN(Salair) , MAX(Salaire)
FROM Employe
GROUP BY NumD ;
- 7) SELECT * FROM Employe
WHERE Salaire > SELECT MAX(Salaire) FROM
Employe ;

Épreuve d'Informatique

Date : 9 Mai 2019

Sections : SM&SP

Durée : 2 H

Nombre de pages : 4

NB : Toutes les fonctions et méthodes doivent être implémentées en python.

Exercice 1 : Évaluation d'un polynôme en un point et Complexité

On veut évaluer $P(x)$, où x est un réel et P est un polynôme de degré n représenté par une liste L . Si $P(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ alors $L = [a_0, a_1, a_2, \dots, a_n]$.

1. Écrire une fonction **eval(L,x)** qui permet de retourner l'évaluation de P en x .
2. Évaluer la complexité de la fonction **eval**.
3. Écrire une fonction **eval_horner(L,x)** qui permet de faire cette évaluation en utilisant la méthode de Horner. La méthode de Horner utilise la factorisation suivante :

$$P(x) = a_0 + x(a_1 + x(a_2 + x(\dots + x(a_{n-1} + xa_n))))$$

4. Évaluer la complexité de la fonction **eval_horner**.
5. Donner la version **réursive** de la fonction **eval_horner**.

Exercice 2 : TOPSIS

La technique TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) est une technique d'aide à la décision multi-critères. L'idée fondamentale de cette technique consiste à choisir parmi un ensemble d'alternatives, la meilleure alternative selon des critères qu'on cherche à maximiser (J) et d'autres critères qu'on cherche à minimiser (J'). L'alternative idéale est la solution qui se rapproche le plus possible (selon une distance) de la solution idéale (A^+) et qui s'éloigne le plus possible de la pire solution (A^-).

Cette technique peut-être formalisée en utilisant une matrice décrivant m alternatives en lignes et n critères en colonnes. La matrice de décision associée est notée X . On note x_{ij} un élément à la ligne i et à la colonne j dans X ($1 \leq i \leq m$ et $1 \leq j \leq n$).

Les étapes de la technique TOPSIS sont décrites comme suit :

- **Étape 1** : Transformer la matrice de décision X en une matrice de décision normalisée X_n , où (n_{ij}) est un élément de la matrice X_n :

$$n_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^m x_{ij}^2}}, 1 \leq i \leq m \text{ et } 1 \leq j \leq n.$$

- **Étape 2** : Transformer la matrice de décision normalisée X_n en une matrice de décision normalisée et pondérée X_p , où (p_{ij}) est un élément de la matrice X_p :

$$p_{ij} = w_j n_{ij}, 1 \leq i \leq m \text{ et } 1 \leq j \leq n.$$

et où w_j est le poids du j -ième critère et $\sum_{j=1}^n w_j = 1$.

- **Étape 3** : La solution idéale (A^+) et la pire solution (A^-) sont ensuite définies comme suit :

$$A^+ = \{p_1^+, \dots, p_n^+\}$$

$$A^+ = \{(\max p_{ij}, j \in J), (\min p_{ij}, j \in J')\}, 1 \leq i \leq m$$

$$A^- = \{p_1^-, \dots, p_n^-\}$$

$$A^- = \{(\min p_{ij}, j \in J), (\max p_{ij}, j \in J')\}, 1 \leq i \leq m$$

- A^+ : minimise les critères qu'on cherche à réduire (J') et maximise les critères qu'on cherche à maximiser (J).
- A^- : minimise les critères qu'on cherche à maximiser (J) et maximise les critères qu'on cherche à réduire (J').
- **Étape 4** : Calculer pour chaque alternative i , la distance euclidienne par rapport à la solution idéale (notée d_i^+) ainsi que la distance euclidienne par rapport à la pire solution (notée d_i^-) :

$$d_i^+ = \sqrt{\sum_{j=1}^n (p_{ij} - p_j^+)^2}, 1 \leq i \leq m$$

$$d_i^- = \sqrt{\sum_{j=1}^n (p_{ij} - p_j^-)^2}, 1 \leq i \leq m$$

- **Étape 5** : Calculer un coefficient de mesure de rapprochement de chaque alternative i par rapport à la solution idéale (notée C_i) :

$$C_i = \frac{d_i^-}{d_i^+ + d_i^-}, 1 \leq i \leq m$$

- **Étape 6** : Ranger les alternatives en fonction des valeurs décroissantes de C_i .
La valeur C_i est entre 0 et 1 : plus cette valeur est proche de 1, plus l'alternative i est meilleure.

Travail demandé :

1. Écrire une fonction **saisir** qui permet de saisir et retourner une matrice de réels de dimension $m \times n$.
2. Écrire une fonction **normaliser** qui permet de normaliser une matrice M telle qu'indiquée à l'étape 1.
NB : On cherche à normaliser la matrice M et non pas à créer une nouvelle matrice normalisée.
3. Étant donnée une liste $W = [w_1, w_2, w_3, \dots, w_n]$ dont les éléments sont des valeurs réelles de somme égale à 1 et où chaque w_j représente le poids du j-ième critère. Écrire une fonction **ponderer** qui permet de pondérer la matrice M telle qu'indiquée à l'étape 2.
NB : On ne vous demande pas de vérifier la somme des poids des critères.
4. Écrire une fonction **Calculer_solutions** qui permet de calculer et retourner la solution idéale (A^+) ainsi que la pire solution (A^-) sous forme de listes python, telle qu'indiquée à l'étape 3. Pour simplifier le problème, nous allons faire l'hypothèse que tous les critères étudiés sont des critères à maximiser : Ainsi, $J^+ = \emptyset$ et la solution (A^+) consiste en l'ensemble des valeurs maximales des critères de l'ensemble J . Tandis que (A^-) est l'ensemble des valeurs minimales des critères de l'ensemble J .
5. À partir de la matrice M et des solutions (A^+) et (A^-), écrire une fonction **Calculer_distances** qui permet de calculer et retourner pour chaque alternative i , les distances (d_i^+) ainsi que la distance d_i^- telle qu'indiquée à l'étape 4. Le résultat doit être sous forme de dictionnaire python, où la clef est l'indice de l'alternative et la valeur est un tuple composé des valeurs (d_i^+ , d_i^-).
6. À partir du dictionnaire contenant les distances (d_i^+ , d_i^-) des différentes alternatives, écrire une fonction **Calculer_Coeff** qui permet de calculer les coefficients de mesure de rapprochement des différentes alternatives par rapport à la solution idéale, telle qu'indiquée à l'étape 5. Le résultat est également un dictionnaire contenant comme clef, l'indice de l'alternative i , la valeur est le coefficient de rapprochement C_i .
7. Écrire une fonction **Classer** qui permet à partir du dictionnaire des coefficients de rapprochement des différentes alternatives par rapport à la solution idéale, de classer (afficher) les différentes alternatives (de la meilleure vers la pire alternative), telle qu'indiquée à l'étape 6.

Exercice 3 : Morpion

Le Morpion est un jeu de réflexion qui se joue à deux et dont le but est de créer le premier alignement (verticalement, horizontalement ou en diagonale) sur une grille. Les deux joueurs inscrivent tour à tour leurs symboles sur la grille, le gagnant est celui qui arrive le premier à l'alignement.

Pour simplifier, nous allons implémenter une version du Morpion en python, en Orienté Objet avec une grille de 3×3 .

Travail demandé :

1. Créer une classe **Morpion** avec son constructeur (**__init__**) définissant les attributs suivants :
 - *jeu* qui représente la grille du jeu. Elle sera initialisée par un tableau numpy carré de taille 3 dont les éléments sont tous égaux au caractère "?".
 - *listeX* qui représente la liste des coordonnées des coups joués par le joueur *X*, et *listeO* qui représente la liste des coordonnées des coups joués par le joueur *O*. Ces attributs seront initialisés par des listes vides et seront remplies au fur et à mesure de l'avancement du jeu.
 - *cases_restantes* qui représente la liste des coordonnées des cases encore vierges. Cet attribut est initialisé par une liste contenant tous les couples (i,j) tels que $0 \leq i,j \leq 2$. Cet attribut sera vidé au fur et à mesure de l'avancement du jeu.
 - Deux attributs de type booléen *gX* et *gO* qui représentent respectivement la victoire/l'échec du joueur *X* et du joueur *O*. Ces attributs sont initialisés par la valeur *False*.
2. Donner la méthode d'affichage (**__repr__**) pour visualiser la grille.
3. Créer une méthode **jouer** qui prend en paramètres *self*, *coord* (tuple contenant les coordonnées d'une case valide, non encore utilisée) et *joueur* (caractère "X" ou "O") et qui modifie :
 - l'attribut *jeu* en remplaçant dans la matrice l'élément "?" situé en ligne et colonne de *coord* par *X* ou *O* selon le paramètre *joueur* ;
 - l'attribut *listeX* ou *listeO* selon le joueur, en ajoutant à la liste correspondante le tuple *coord* ;
 - l'attribut *cases_restantes* en supprimant le tuple *coord* de cette liste.

Indication : On utilisera la méthode `L.remove(obj)` qui supprime la première occurrence de *obj* dans une liste *L* ;

4. Écrire une méthode **tester** qui teste si trois 'X' ou trois 'O' sont alignés et modifie l'attribut *gX* ou *gO* lorsque c'est le cas en lui donnant la valeur *True*.

Indication : Pour cela, nous utiliserons :

- Une variable globale, une liste *L* contenant les listes des coordonnées **des différentes possibilités d'alignements de trois cases**. Par exemple, *L* peut-être définie de la manière suivante :

$$L = [[(0,j) \text{ for } j \text{ in range}(3)], [(1,j) \text{ for } j \text{ in range}(3)], [(2,j) \text{ for } j \text{ in range}(3)], [(i,0) \text{ for } i \text{ in range}(3)], [(i,1) \text{ for } i \text{ in range}(3)], [(i,2) \text{ for } i \text{ in range}(3)], [(i,i) \text{ for } i \text{ in range}(3)], [(i,3-i-1) \text{ for } i \text{ in range}(3)]]$$
 - Une fonction supposée prédéfinie intitulée `nb_commun(L1,L2)` qui prend deux listes en argument et renvoie le nombre d'éléments qu'elles ont en commun.
5. Écrire une méthode **Jouer_Ordi** permettant de jouer contre l'ordinateur. L'utilisateur est le joueur *X* et l'ordinateur est le joueur *O*. Quand c'est son tour, le joueur doit saisir les coordonnées d'une case. Il faudra faire un contrôle à la saisie pour que le joueur n'introduise pas des coordonnées invalides ou des coordonnées d'une case déjà remplie. Quand c'est le tour de l'ordinateur, ses coordonnées sont choisies aléatoirement parmi les cases restantes. A la fin du jeu, on affichera au joueur "Bravo, vous avez gagné" ou bien "Vous avez perdu".

Indication : on utilisera la fonction `choice` du package `random`. `choice(seq)` qui retourne un élément aléatoire à partir d'une séquence *seq*.

Correction de l'examen informatique N°2**Mai 2019****IPEIT-SM&SP**

Exercice1 :

Exercice 2 :

Exercice 3: Morpion

Exercice 1 (10 pts)

1.

```
def eval(L,x):  
    s=0  
    for i in range(len(L)):  
        s=s+L[i]*x**i  
    return(s)
```
2. Complexité de la fonction eval : $O(n^{**2})$
3.

```
def eval_horner(L,x):  
    s=L[len(L)-1]  
    for i in range(len(L)-2,-1,-1):  
        s=s*x+L[i]  
    return(s)
```
4. Complexité de la fonction eval_horner : $O(n)$
5.

```
def horner_rec(L,x):  
    if len(L)==1:  
        return L[0]  
    else:  
        return L[0]+x*horner_rec(L[1:],x)
```

Exercice 2 :

import numpy as np

1.

```
def saisir(m,n):  
    M=np.zeros((m,n))  
    for i in range(m):  
        for j in range(n):  
            M[i,j]=float(input("donner un reel"))  
    return M
```
2.

```
def normaliser(M):  
    for j in range(M.shape[1]):  
        v=np.sqrt(sum(M[:,j]**2))  
        for i in range(M.shape[0]):  
            M[i,j]=M[i,j]/v
```
3.

```
def ponderer(M,w):  
    for j in range(M.shape[1]):  
        M[:,j]=M[:,j]*w[j]
```
4.

```
def calculer_solutions(M):  
    return list(M.max(axis=0)),list(M.min(axis=0))
```
5.

```
def calculer_solutions_version2(M):  
    a1=[]  
    a2=[]  
    for i in range(M.shape[1]):  
        a1.append(M[:,i].max())
```

```
        a2.append(M[:,i].min())
    return a1,a2
6. def calculer_distance(M,a1,a2):
    d=dict()
    for i in range(M.shape[0]):
        s1,s2=0,0
        for j in range(M.shape[1]):
            s1=s1+(M[i,j]-a1[j])**2
            s2=s2+(M[i,j]-a2[j])**2
        d[i+1]=(np.sqrt(s1),np.sqrt(s2))
    return d
7. def Calculer_Coeff(d):
    d2=dict()
    for i in d:
        d2[i]=d[i][1]/(d[i][1]+d[i][0])
    return d2
8. def classement(c):
    L=list()
    for i in c:
        L.append((i,c[i]))
    while len(L)!=0:
        max=0
        for i in range(len(L)):
            if L[i][1]>=max:
                max=L[i][1]
                al=L[i][0]
        print(al,max)
        pos=L.index((al,max))
        L.pop(pos)
#-----Test -----Non demandé
M=saisir(4,5)
print(M)
print("resultat de la normalisation ")
normaliser(M)
print(M)
ponderer(M,[0.2,0.2,0.2,0.2,0.2])
print("apres ponderation ", M)
t=calculer_solutions(M)
print("solution A+ ",t[0])
print("solution A- ",t[1])
d=calculer_distance(M,t[0],t[1])
print("distances ",d)
c=Calculer_Coeff(d)
print("closness ",c)
classement(c)
```

Exercice 3 : Morpion

```
import numpy as np
import random as r
```

Travail demande

class Morpion:

```
    def __init__(self):
        self.jeu=np.array([[ '?','?','?'],[ '?','?','?'],[ '?','?','?']])
        self.listeX=[]
        self.listeO=[]
        self.cases_restantes=[(i,j) for i in range(3) for j in range(3)]
        self.gX=False
        self.gO=False
    def __repr__(self):
        return(str(self.jeu))
    def joue(self,coord,joueur):
        if joueur=='X':
            self.jeu[coord[0],coord[1]]='X'
            self.listeX.append(coord)
        else:
            self.jeu[coord[0],coord[1]]='O'
            self.listeO.append(coord)
            self.cases_restantes.remove(coord)
    def teste(self):
        for liste in LG:
            if nb_commun(liste,self.listeX)==3:
                self.gX=True
            elif nb_commun(liste,self.listeO)==3:
                self.gO=True
    def jeu_contre_ordi(self):
        while self.gX!=True and self.gO!=True and len(self.cases_restantes)>0:
            print(self.jeu)
            while True:
                l=int(input('donner num ligne'))
                c=int(input('donner num colonne'))
                if (l,c) in self.cases_restantes:
                    self.joue((l,c),'X')
                    self.teste()
                    break
            else:
                if l>=3 or c>=3:
                    print('Coordonnees invalides, refaire la saisie!!!')
                else:
                    print('Case deja remplie, refaire la saisie!!!')
            if self.gX==False:
                d=r.choice(self.cases_restantes)
                self.joue(d,'O')
                self.teste()
        if self.gX==True:
            return ('Bravo, vous avez gagne!!!')
        else:
            return('Vous avez perdu!!!')
```

#-----Test-----Non demandé

```
LG=[[ (0,j) for j in range(3)],  
      [(1,j) for j in range(3)],  
      [(2,j) for j in range(3)],  
      [(i,0) for i in range(3)],  
      [(i,1) for i in range(3)],  
      [(i,2) for i in range(3)],  
      [(i,i) for i in range(3)],  
      [(i,3-i-1) for i in range(3)]]  
def nb_commun(L1,L2):  
    nb=0  
    for i in L1:  
        if i in L2:  
            nb=nb+1  
    return nb  
  
M=Morpion()  
print(M.jeu_contre_ordi())
```

Examen Informatique N°1

Sections : SM & SP

Documents non autorisés

Date : Décembre 2019

Durée : 2h

Nombre de Pages : 5

Problème 1 (10 points)

Partie 1

Dans ce problème, nous désirons modéliser un réseau social dans lequel plusieurs utilisateurs identifiés par des numéros 0, 1, 2, 3, . . . sont inscrits.

Pour ce faire, nous allons implémenter la classe **reseauSocial** caractérisée par un attribut intitulé **data**, représenté par un dictionnaire, où les clefs du dictionnaire sont les identifiants (entier) des utilisateurs du réseau social. Les valeurs du dictionnaire sont des listes contenant des valeurs 0 et 1, représentant les liens d'amitié entre les différents utilisateurs.

Où chaque valeur **data[i][j]** vaut :

— 1 si l'utilisateur **i** est ami avec l'utilisateur **j**

— 0 sinon.

Notez bien qu'un utilisateur n'est jamais ami avec lui-même et que pour chaque couple d'utilisateurs **i** et **j** : si **i** est l'ami de **j** alors **j** est aussi l'ami de **i**.

Par exemple, un réseau social dans lequel il y a trois utilisateurs, où l'utilisateur 0 est ami avec les deux utilisateurs 1 et 2 et où les deux utilisateurs 1 et 2 ne sont pas amis, peut-être défini par l'attribut **data** comme suit :

Clefs: Identifiants utilisateur	Valeurs : Liste d'amis
0	[0,1,1]
1	[1,0,0]
2	[1,0,0]

Rappel

On rappelle les opérations utiles sur un dictionnaire d :

d[i] retourne la valeur associée à la clef **i** du dictionnaire **d**. Si la clef **i** n'existe pas dans **d**, cette syntaxe génère une erreur.

d.keys() retourne un itérable formé par les clefs du dictionnaire **d**.

d.values() retourne un itérable formé par les valeurs du dictionnaire **d**.

d.items() retourne un itérable formé par des couples (k,v) où k est une clef du dictionnaire et v est la valeur associée.

Travail demandé :

Définir la classe **reseauSocial** avec les méthodes suivantes :

1) Le constructeur de la classe **__init__(self,n)** qui permet d'initialiser l'attribut **data** du réseau social de **n** utilisateurs identifiés par des entiers de **0** à **n-1** tel qu'il n'existe aucune relation d'amitié entre les différents utilisateurs.

2) La méthode d'affichage spéciale **__repr__(self)** qui permet d'afficher un réseau social de la façon suivante :

0:[0, 1, 1]

1:[1, 0, 0]

2:[1, 0, 0]

3) La méthode **ajouterLien(self,i,j)** qui permet d'ajouter un lien d'amitié entre deux utilisateurs **i** et **j**. Cette méthode :

- retourne la valeur 0 dans le cas où le lien d'amitié ne peut pas être créé (**i=j**) ou dans le cas où l'un des utilisateurs n'est pas dans le réseau.
- sinon, cette méthode crée un lien d'amitié entre les deux utilisateurs **i** et **j** et retourne la valeur 1.

4) La méthode **nombreAmis(self,i)** qui permet de retourner le nombre d'amis d'un utilisateur **i**. Si l'utilisateur **i** n'existe pas dans le réseau, cette méthode retourne la valeur -1.

5) La méthode **retournerAmis(self,i)** qui permet de retourner la liste des amis de l'utilisateur **i**. Cette méthode retourne une liste vide si l'utilisateur **i** n'est pas un utilisateur du réseau ou qu'il ne possède pas d'amis.

6) La méthode **ajouterIndividu(self)** qui permet d'ajouter un individu au réseau en lui attribuant automatiquement un identifiant à partir du dernier identifiant des utilisateurs du réseau. Cette méthode doit donc :

- ajouter l'individu au réseau des utilisateurs
- lui attribuer une liste d'amis de telle sorte que l'individu ajouté n'a aucun lien d'amitié avec les différents utilisateurs du réseau.
- retourner le nouvel identifiant attribué au nouvel individu.

NB : on n'oubliera pas également de faire la mise à jour de la liste d'amis des différents utilisateurs du réseau pour la prise en considération de ce nouvel individu.

Partie2

Dans cette deuxième partie, on va implémenter une nouvelle classe **reseauSocialComplet** qui dérive de la classe **reseauSocial** en ajoutant en plus l'information sur les centres d'intérêts des différents utilisateurs du réseau. La classe **reseauSocialComplet** est donc caractérisée par l'attribut **data** tel que défini dans la classe **reseauSocial**.

Les centres d'intérêts sont modélisés par un attribut intitulé **interets**. Cet attribut est représenté par une liste de listes, où chaque sous-liste à la position **i** indique les centres d'intérêts (chaînes de caractères) de l'utilisateur **i** du réseau.

Travail demandé :

Définir la classe **reseauSocialComplet** avec les méthodes suivantes :

- 1) Le constructeur de la classe **__init__(self,n,L)** qui permet d'initialiser un réseau social complet de **n** utilisateurs identifiés par des entiers de **0** à **n-1** tel qu'il n'existe aucune relation d'amitié entre les différents utilisateurs. Cette méthode doit également définir l'attribut **interets** à partir de la liste L (liste de listes des intérêts des utilisateurs).
- 2) La méthode d'affichage spéciale **__repr__(self)** qui permet d'afficher un réseau social complet de la façon suivante :
0:[0, 1, 1]:musique sport
1:[1, 0, 0]:musique lecture
2:[1, 0, 0]:politique peinture
- 3) La méthode **ajouterIndividu(self)** qui permet d'ajouter un individu au réseau social tel que défini à la **question 6 – Partie 1** et de lui attribuer une liste de centre d'intérêt vide.
- 4) La méthode **ajouterInteret(self,i,j)** qui permet d'ajouter un centre d'intérêt **j** à l'utilisateur **i** du réseau et de retourner 1. Si l'utilisateur **i** n'existe pas dans le réseau, la méthode retourne 0.
- 5) La méthode **retournerAmisCentres(self,i)** qui permet de retourner une liste des amis de l'utilisateur de **i** qui ont au moins un centre d'intérêt commun avec l'utilisateur **i**.

Problème 2 (10 points)

Une société de chemin de fer souhaite mettre en place une application en Python permettant d'automatiser la gestion de sa clientèle ainsi que les tarifs des titres de transport utilisés.

Dans le système on distingue trois types de clients : les **Passagers**, les Personnes ayant un Handicap « **PAH** », et les **Abonnés**.

Chaque client est caractérisé par les attributs:

- **cin** : de type chaîne de caractère.
- **nom** : de type chaîne de caractère.
- **age** : de type entier.

Une Personne ayant un Handicap « PAH » est un client caractérisé en plus par l'attribut:

- **accompagné** : de type booléen qui indique s'il est accompagné par une autre personne ou non.

Un abonné est un client caractérisé en plus par l'attribut:

- **catégorie** : de type chaîne de caractère désignant la catégorie de l'abonné.

Un client qui n'est ni un « PAH » ni un abonné est un Passager, qui paie un ticket dont le tarif dépend de son âge.

La valeur du tarif du ticket est définie en fonction du type du client et d'un tarif de base qui est égal à **0.900**, de la manière suivante :

Type du Client	Formule de calcul du tarif
PAH	Si La « PAH » est accompagnée alors $\text{tarif} = 0.2 * \text{tarif de base}$, sinon $\text{tarif}=0$
Passager	Si $\text{âge} \leq 3$ alors $\text{tarif}=0$, Si $3 < \text{âge} \leq 7$ alors $\text{tarif} 0.3 * \text{tarif de base}$, Sinon $\text{tarif} = \text{le tarif de base}$

Le calcul du montant de l'abonnement d'un client abonné est défini en fonction de la catégorie. Sachant que la catégorie peut être soit « Etudiant » soit « Non Etudiant », le montant d'abonnement est calculé selon la règle suivante:

Si la catégorie de l'abonné est « Etudiant » alors le montant de l'abonnement est égal à 47 sinon il est égal à 70.

Partie 1

- 1) Définir la classe **Client**, possédant:
 - a) Un attribut de classe **TARIF_BASE** égal à 0.900,
 - b) Un **constructeur**,
 - c) Une **méthode de représentation** du client permettant l'affichage d'un client selon le format : Client (08132456, Imen, 32 ans).
- 2) Réaliser la classe **PAH** dérivée de la classe **Client** et possédant :
 - a) Un **constructeur**,
 - b) Une **méthode de représentation** de la PAH permettant l'affichage des propriétés en respectant le format :
Client (05469182, Majdi, 45 ans, accompagné),
Ou Client (05214563, Mohamed, 15 ans, non accompagné),
 - c) Une méthode **calculTarif** permettant de calculer le tarif du ticket d'une PAH.
- 3) Réaliser la classe **Passager** dérivée de la classe **Client** et possédant :
 - a) Un **constructeur** si nécessaire,
 - b) Une méthode **calculTarif** permettant de calculer le tarif du ticket d'un Passager.
- 4) Réaliser la classe **Abonne** dérivée de la classe **Client** et possédant :
 - a) Un **constructeur**,
 - b) Une **méthode de représentation** de l'abonné permettant l'affichage des propriétés en respectant le format : Client (07663215, Amine, 20 ans, Etudiant),
 - c) Une méthode **calculAbonnement** permettant de calculer la valeur de l'abonnement.

Partie 2

On se propose maintenant de réaliser une classe **Wagon** désignant un wagon du train et caractérisée par :

- **code** : de type entier.
- **clients** : liste des clients qui sont à bord du wagon.
- **cpMax** : de type entier représentant la capacité maximale du wagon.

Définir la classe **Wagon** fournissant :

- 1) Un **Constructeur paramétré** par le code et la capacité maximale du wagon. Il initialise la liste des clients à une liste vide.
- 2) Une **méthode de représentation** du wagon permettant d'afficher le code du wagon ainsi que la description de tous les clients qui sont à bord du wagon comme suit :
Wagon N° 12452 de capacité maximale 42
Clients à bord:
Client (08132456, Imen, 32 ans)
Client (05469182, Majdi, 45 ans, accompagné)
Client (07663215, Amine, 20 ans, Etudiant)
- 3) Une méthode **monterClient** permettant d'ajouter le client passé en paramètre à la liste des clients à bord du wagon.
NB : prévoir le cas de dépassement de capacité.
- 4) Une méthode **recette** permettant de renvoyer le total des tarifs des tickets de tous les clients qui sont à bord du wagon. Les abonnements ne seront pas considérés.
NB : on rappelle la fonction prédéfinie **type(x)** qui retourne le type d'une variable x donnée.
- 5) Une méthode **lesPAH** qui retourne la liste des personnes ayant un handicap qui sont à bord du wagon.
- 6) Une méthode **totalParCatégorie** qui retourne le nombre total des abonnés de la catégorie passée en paramètre qui sont à bord du wagon.

Correction de l'examen informatique N°1**Décembre 2019****IPEIT-SM&SP****Problème 1 (10 pts)**

#partie1 (5.5pts)

class reseau:

```
#1) def __init__(self,n):
    self.data={}
    for i in range(n):
        L=[0 for j in range(n)]
        self.data[i]=L
#2) def __repr__(self):
    ch=""
    for i,j in self.data.items():
        ch=ch+str(i)+':'+str(j)+'\n'
    return ch
#3) def ajout_lien(self,i,j):
    if i==j: return 0
    elif i not in self.data or j not in self.data: return 0
    else:
        self.data[i][j]=1
        self.data[j][i]=1
        return 1
#4) def nb_amis(self,i):
    if i not in self.data: return -1
    else: return (self.data[i].count(1))

#5) def retourner_amis(self,i):
    if i in self.data:
        amis=[]
        for j in range(len(self.data[i])):
            if self.data[i][j]==1: amis.append(j)
        return amis
    else: return[]

#6) def ajouter_individu(self):
    for i in self.data:
        self.data[i].append(0)
    clef_max=max(self.data.keys())
    L=[0 for i in range(len(self.data)+1)]
    self.data[clef_max+1]=L
    return clef_max+1
```

#partie2 (4.5pts)

class reseau_interet(reseau):

```
#1) def __init__(self,n,L):
```

```
        reseau.__init__(self,n)    #ou super().__init__(n)
        self.interets=L

#2) def __repr__(self):
    ch=""
    for i,j in self.data.items():
        ch=ch+str(i)+':'+str(j)+':'
        for k in self.interets[i]:
            ch=ch+k+' '
        ch=ch+'\n'
    return ch

#3) def ajouter_individu(self):
    c=reseau.ajouter_individu(self)
    self.interets.append([])
    return c

#4) def ajouter_interet(self,i,j):
    if i in self.data:
        self.interets[i].append(j)
        return 1
    else: return 0

#5) def retourner_amis_centres(self,i):
    centre=self.interets[i]
    print('centres inte de ',i,centre)
    amis=reseau.retourner_amis(self,i)
    print('les amis',amis)
    LL=list()
    for a in amis:
        for j in self.interets[a]:
            if j in centre:
                print(j)
                LL.append(a)
    return LL
    LL=[a for a in amis for j in self.interets[a] if j in centre]
    return LL

#programme principal non demandé
r=reseau(3)
print(r)
print(r.ajout_lien(0,2))
print(r.ajout_lien(1,2))
print(r.ajout_lien(1,1))
print(r.ajout_lien(1,5))
print(r)
print(r.nb_amis(0))
print(r.retourner_amis(0))
c=r.ajouter_individu()
r.ajout_lien(c,1)
```

```
print('ici')
print(r)

L=[['musique','sport'],['musique','lecture'],['politique','peinture']]
rr=reseau_interet(3,L)
print(rr)
print(rr.ajout_lien(1,2))
print(rr)
print(rr.nb_amis(2))
print(rr.retourner_amis(3))
c=rr.ajouter_individu()
print(c)
print(rr)
print(rr.ajout_lien(1,c))
print(rr.ajout_lien(2,c))
print(rr.retourner_amis(3))
rr.ajouter_interet(c,'equitation')
rr.ajouter_interet(c,'velo')
print(rr)
rr.ajouter_interet(2,'musique')
rr.ajouter_interet(3,'lecture')
print(rr)
print(rr.retourner_amis_centres(1))
```

Problème2 (10 pts)

#Partie 1 (5.5pt)

#1) class client:

#a)

tarif_base=0.9

#b)

def __init__(self,cin,nom,age):

self.cin=cin

self.nom=nom

self.age=age

#c)

def __repr__(self):

return "Client (" + self.cin + "," + self.nom + "," + str(self.age) + "ans)"

#2)

class PAH (client):

#a)

def __init__(self,cin,nom,age,accompagne):

self.accompagne=accompagne

super().__init__(cin,nom,age)

#ou bien client.__init__(self,cin,nom,age)

#b)

def __repr__(self):

ch=super().__repr__()

ch=ch[:len(ch)-1]

if self.accompagne==True:

ch=ch+", accompagné"

```
        else:
            ch=ch+",non accompagné)"
        return ch

#c)
def calculTarif(self):
    if self.accompagne:
        return self.tarif_base*0.2
    else: return 0

#3)
class Passager(client):
    #a)
    # constructeur non nécessaire
    #ou bien
    def __init__(self,cin,nom,age):
        super().__init__(cin,nom,age)

    #b)
    def calculTarif(self):
        if self.age <=3 : return 0
        elif self.age <= 7 : return 0.3 * self.tarif_base
        else: return self.tarif_base

#4)
class Abonne(client):
    #a)
    def __init__(self,cin,nom,age,categorie):
        self.categorie=categorie
        super().__init__(cin,nom,age)
    #b)
    def __repr__(self):
        ch=super().__repr__()
        ch=ch[:len(ch)-1] + "," + self.categorie + ")"
        return ch
    #c)
    def calculAbonnement(self):
        if self.categorie=="Etudiant":
            return 47
        else:
            return 70

#Partie 2 (4.5pt)
class Wagon:
#1)  def __init__(self,code,cpMax):
        self.code=code
        self.clients=[]
        self.cpMax=cpMax

#2)  def __repr__(self):
```

```
        ch="Wagon N°" + str(self.code) + "de capacité maximale" + str(self.cpMax) + "\n" +  
"Clients à bord:" + "\n"  
        for c in self.clients:  
            ch=ch+c.__repr__()+"\n"  
        return ch
```

#3)

```
def monterClient(self,c):  
    if len(self.clients) < self.cpMax:  
        self.clients.append(c)  
    else:  
        print("impossible : capacité maximale atteinte")
```

#4)

```
def recette(self):  
    s=0  
    for c in self.clients:  
        if type(c)!=Abonne:  
            s+=c.calculTarif()  
    return s
```

#5)

```
def lesPAH(self):  
    return [c for c in self.clients if type(c)==PAH]
```

#6)

```
def totalParCatégorie(self,cat):  
    return len ([c for c in self.clients if type(c)==Abonne and c.categorie==cat])
```

#programme principal non demandé

```
c1=client('12','imen',30)  
print(c1)  
c2=Passager('5','ahmed',70)  
print(c2)  
c3=PAH('6','mohamed',10,True)  
print(c3)  
c4=PAH('6','imed',10,False)  
print(c4)  
c5=Abonne("54698","emna",23,"Etudiant")  
print(c5)  
c6=Abonne("54698","youssef",23,"Non Etudiant")  
print(c6)  
w=Wagon(12,42)  
print(w)  
for c in [c2,c3,c4,c5,c6]:  
    w.monterClient(c)  
print(w)  
s=w.recette()  
print(s,"DT")  
print(w.lesPAH())  
print(w.totalParCatégorie("Etudiant"))  
print(w.totalParCatégorie("Non Etudiant"))
```

DS Informatique N°2

Classes : 2^{ème} année SP/SM

Durée : 2h

Nb. Pages : 3

N.B : Les exercices sont indépendants ainsi que les deux parties de l'exercice 2.

Exercice 1

Un chef cuisinier s'est constitué une base de données sur les personnes qu'il invite et les plats qu'il leur sert. On se propose de traiter en partie cette base de données définie par les trois tables suivantes :

Repas(date, invité)

Menu(date, nom_plat)

Preference(nom_personne, nom_plat)

- La table Repas donne la liste des invités qui ont été reçus et à quelle date.
- La table Menu donne le menu servi à chaque date
- La table Preference donne pour chaque personne ses plats préférés
- Les attributs "nom_personne" et "invité" ont même domaine.
- L'attribut date est une chaîne de caractères.

Écrire en algèbre relationnelle les requêtes suivantes :

1. Quels sont les invités du repas du "120319".
2. Quels sont les plats qui ont été servis à Sami ?
3. Quels sont les invités qui, à un repas, ont eu au moins un de leurs plats préférés ?
4. Quels sont les invités qui n'ont jamais eu un de leur plat préféré ?
5. Quels sont les plats qui ont été servis dans au moins deux menus différents ?
6. Quelles sont les personnes qui n'ont jamais été invités ?
7. Quels sont les invités qui sont venus à tous les repas ?

Exercice 2

On propose de gérer en partie le système d'information géographique d'un pays X afin de localiser l'information et de maintenir une organisation plus conviviale des données.

Soit le schéma relationnel de la base de données, '**geographie.sq3**', du pays X définie par les trois relations suivantes :

Region(NumRegion, nom)

Département(NumDept, *NumRegion*, nom)

Commune(NumCom, *NumDept*, nom, surface, pop17, pop18, zmin, zmax)

- La table **Region** contient toutes les régions du pays.
- La table **Département** contient tous les départements de la région. Un département appartient à une seule région.
- La table **commune** contient toutes les communes du pays.
 - zmin et zmax sont respectivement les altitudes minimales et maximales en mètres.
 - pop17 et pop18 sont respectivement la population en 2017 et celle du 2018.

Partie 1 : SQLite

Soit le script Python suivant :

```
import sqlite3
con=sqlite3.connect('geographie.sq3')
cur=con.cursor()
...
cur.close()
con.close()
```

Dans la suite, con et cur seront utilisés comme des variables globales.

1. Écrire la fonction Python **CreerTable**, qui permet de créer la table **Département** dans la base en exprimant toutes les contraintes d'intégrité.
2. Écrire la fonction Python **GetData**, qui, à partir d'une table T, détermine et retourne la liste de toutes les informations des lignes de cette table.
3. Écrire la fonction Python **RechCommune**, qui, à partir de deux réels z1 et z2 :
 - détermine la liste L de toutes les informations de la table commune.
 - détermine, à partir de L, la liste des numéros communes dont l'écart entre les altitudes (en mètres) est comprise entre z1 et z2.
4. On dispose d'un fichier 'communes.txt' en format texte contenant toutes les communes du pays X. Chaque ligne du fichier contient les informations, relatives à une commune donnée, séparées par le caractère '#' et débute par un espace.
Exemple : La ligne, ' C01#52#Ar#2500#12000#13000#250#500', représente la commune Ar, identifiée par 'C01', ayant comme surface 2500 et respectivement pop17 et pop18 12000 et 13000. zmin et zmax ont respectivement les valeurs 250 et 500.
 - Écrire la fonction Python **Remplir**, qui à partir du fichier 'communes.txt', remplit la table commune.
 - Écrire la fonction Python **Afficher**, permettant d'afficher toutes les informations relatives aux communes.

Partie 2 : SQL

Dans la suite, on suppose que les trois tables sont créées et remplies.

Donner les requêtes SQL permettant de répondre aux questions suivantes :

1. Combien d'habitants en 2018 ? (en supposant que chacun n'a été compté qu'une fois dans les différentes communes. . .).
2. Combien de communes dans le département 10 ?
3. Quelles sont les dix plus petites communes, en superficie ?
4. Quelles sont les dix communes les plus peuplées ?
5. Donner pour chaque département, le nombre de communes.
6. Donner les numéros de département, avec pour chaque numéro la population totale du département. Trier par population totale décroissante et limiter la liste aux départements ayant plus d'un million d'habitants.
7. Quelles sont les quinze communes pour lesquelles l'écart entre les altitudes maximale et minimale est le plus grand ?
8. Donner les noms des départements des régions "B" et "P".
9. Donner les noms de départements, avec pour chaque département le nombre de communes. Ordonner par population décroissante.
10. Donner les noms des régions avec la densité de population de chaque région.
11. Donner les noms de communes dont la population excède 100 fois la population moyenne des différentes communes.
12. Donner les noms de communes dont l'altitude minimale est la deuxième plus petite valeur parmi celles qui sont au moins égales à 250.
13. Donner sans doublons les noms des régions contenant au moins un département dont le nom commence par "V".
14. Donner les noms des régions avec pour chacune le nom et la population de la commune la plus peuplée de la région.

Bon travail

Corrigé du DS Informatique N°2

Mars 2019

IPEIT - SM&SP

Exercice 1

1. $\pi_{\text{invité}}(\sigma_{\text{date}='120319'}(\text{Repas}))$
2. $\pi_{\text{nomPlat}}(\sigma_{\text{invite}='Sami'}(\text{Plat} \bowtie_{\text{plat.date}=\text{Repas.date}} \text{Repas}))$
3. $R1 = \text{Preferences} \bowtie_{\text{Preferences.nomplat}=\text{Plat.nomplat}} \text{Plat} \bowtie_{\text{Plat.date}=\text{Repas.date}} \text{Repas}$
 $R2 = \sigma_{\text{invite}=\text{personne}}(R1)$
 $R3 = \pi_{\text{invite}}(R2)$
4. $\pi_{\text{invite}}(\text{Repas}) - R3$
5. $\pi_{\text{Personnes}}(\text{Preferences}) - \pi_{\text{invite}}(\text{Repas})$
6. $\text{Repas} \div \pi_{\text{date}}(\text{Repas})$

Exercice 2 :

Partie 1

1. **def Creer_table() :**
`cur.execute('create table if not exists Gouvernorat(Idgouv text primary key,
IdReg text foreign key references (Region),
nomG text);')`
2. **def Creer_liste(T) :**
`cur.execute('select * from '+T)
L=cur.fetchall()
return L`
3. **def RechMunicip(z1, z2) :**
`L= Creer_liste('Municipalite')
L_id= []
for m in L :
ecart= m[6]-m[5]
if z1 <= ecart <= z2:
L_id.append(m[0])
return L_id`
4. **def Remplir(nom_fichier='Municipalite.txt'):**
`f= open(nom_fichier, 'r')
L=f.readlines()
for i in L
ch= i[: len(i)-1] # ou bien i.strip('\n')
L1=ch.split('#')
cur.execute(' insert into Municipalite values (?, ?, ?, ?, ?, ?, ?)',
(L1[0], L1[1], L1[2], float(L1[3]), float(L1[4]), float(L1[5]), float(L1[6]))
f.close()
con.commit()`

5. **def Afficher():**

```
cur.execute( 'select * from Municipalite where pop18> 30000')
L=cur.fetchall()
for i in L:
    print( i )
```

Partie 2:

1. select sum(pop18) from Municipalite ;
2. select count(IdMunni) from Municipalite where IdGouv= 'G12' ;
3. select nom from Municipalite where surface= (select min(surface) from Municipalite) ;
4. select count(*), IdGouv from Municipalite group by IdGouv ;
5. select IdGouv, sum(pop18) as TotPop from Municipalite
group by IdGouv having TotPop >1000000
order by TotPop desc;
6. select G.nomG from Gouvernorat G, Region R
where G.IdReg=R.IdReg
and R.nomR in ('B', 'P') ;
7. select R.nomR, sum(M.pop18)/sum(M.surface) densite
from Region R, Gouvernorat G, Municipalite M
where R.IdReg=G.IdReg and M.IdGouv=G.IdGouv
group by R.IdReg ;
8. select nom, pop18 from Municipalite where
pop18 >100* (select avg(pop18) from Municipalite) ;
9. select distinct R.nomR from Region R, Gouvernorat G
where R.IdReg =G.IdReg and G.nomG like 'v%' ;
10. select R.nomR, M.nom, M.pop18 from Region R, Gouvernorat G, Municipalite M
where R.IdReg=G.IdReg and M.IdGouv=G.IdGouv
group by R.nomR having M.pop18=max(M.pop18) ;
11. update Municipalite set surface = 3000 where IdMuni='M03' ;