REPUBLIQUE TUNISIENNE
Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique

Concours Nationaux d'Entrée
aux Cycles de Formation d'Ingénieurs
Session 2018

الجمهورية التونسية
وزارة التعليم العالي والبحث العلمي

المناظرات الوطنية للدخول
إلى مراحل تكوين المهندسين
دورة 2018

Altérative de Correction
Concours Mathématique et Physique, Physique et Chimie et Technologie
Epreuve d'Informatique

PROBLEME 1

Partie I

```
import numpy as np
from random import randint
```

1.

```
def GenererMat(n,m):
    M=np.ndarray(shape=(n,m),dtype=int)
    #ou M=np.empty((n,m),dtype=int)
    for i in range(n):
        for j in range(m):
            M[i,j]= randint(0,255)
             # ou encore np.random.randint (0,256)
    return M
```

2.

```
def MoyVoisinage(M):
    V= M.copy()
    (n,m)=M.shape #ou (n,m)=np.shape(M)
    for i in range(1,n-1):
        for j in range(1,m-1):
            V[i,j]=sum([M[k,l] for k in range(i-1,i+2) \
                    for l in range(j-1,j+2)]) // 9
    return V
```

3.

```
def NbPixels(M,V,i,j):
    (n,m)=M.shape #ou (n,m)=np.shape(M)
    nb=0
    for a in range(n):
        for b in range(m):
            if M[a,b]==i and V[a,b]==j:
                nb+=1
    return nb
```

4.

```
def Histogramme (M,V):
    H=np.zeros((256,256))
    for i in range(256):
        for j in range(256):
            H[i,j]=NbPixels(M,V,i,j)
    return H
```

5.

```
def Sigma(H,binf1,binf2,bsup1,bsup2,comp1=0):
    if not comp1:
        return sum([i*H[i,j] for i in range(binf1,bsup1+1) \
              for j in range(binf2,bsup2+1)])
    else:
        return sum([j*H[i,j] for i in range(binf1,bsup1+1) \
              for j in range(binf2,bsup2+1)])
```

6.

```
def seuillage(H):

    Moy=np.array([[Sigma(H,0,0,256,256)],[Sigma(H,0,0,256,256,1)]
    ])
    varmax=-1
    for s in range(255):
        for t in range(255):
            c=256*256
            P0=sum([H[i,j] for i in range(s+1)\
                   for j in range(t+1)])/c
            P1=sum([H[i,j] for i in range(s+1,256) \
                  for j in range(t+1,256)])/c
            if P0!=0.0 and  P1!=0.0:
                Moy0=(1/P0)*np.array([[Sigma(H,0,0,s+1,t+1)],\
                  [Sigma(H,0,0,s+1,t+1,1)]])
                Moy1=(1/P1)*np.array([[Sigma(H,s+1,t+1,255,255)],
                \
                  [Sigma(H,s+1,t+1,255,255,1)]])
                A=Moy0-Moy
                B=Moy1-Moy
                S=P0*np.dot(A,np.transpose(A))+ \
                  P1*np.dot(B,np.transpose(B))
                var=np.trace(S)
                if varmax<var:
                    varmax=var
                    stMax=(s,t)
    return stMax
```

Partie II

```
#Réponse question 1
class Pixel:
    def __init__(self,a,b,c):
        self.posi=a
        self.posj=b
        self.ton=c
    def __str__(self):
        return   'Pixel   <{},{},{}>'.format(self.posi,self.posj,\
                  self.ton)
        # ou encore return
        #'Pixel <'+str(self.posi)+','+str(self.posj)+ \
            ','+str(self.ton)+'>'

class Region:
    def __init__(self,lab):
        self.label=lab
        self.dict_pixel={}

#Réponse question 2
    def __len__(self):
        nb=0
        for ton in self.dict_pixel: # ou self.dict_pixel.keys()
            nb+=len(self.dict_pixel[ton])
        return nb

#Réponse question 3
    def __call__(self,ton):
       try :
            assert(ton in self.dict_pixel)
            return self.dict_pixel[ton]
       except:
            return None

#Réponse question 4
    def __contains__(self,px):
        #version 1
        if px.ton in self.dict_pixel:
            if (px.posi,px.posj) in self.dict_pixel[px.ton]:
                return True
            else : return False
        else : return False

#Réponse question 5
    def ajouter_pixel(self,px):
        #version1
        if px.ton in self.dict_pixel:
            self.dict_pixel[px.ton].append((px.posi,px.posj))
        else:
            self.dict_pixel[px.ton]=[(px.posi,px.posj)]
```

```
#Réponse question 6
    def supprimer_pixel(self,px):
        if px.ton in self.dict_pixel:
            if len(self(px.ton)) == 1:
                self.dict_pixel.pop(px.ton)
            else :
                self.dict_pixel[px.ton].remove((px.posi,px.posj))
        else :  print("le pixel n'existe pas")

#Réponse question 7
    def binariser_reg(self,seuil):
        other=Region(self.label+'bin')
        other.dict_pixel[0]=[]
        other.dict_pixel[255]=[]
        for i in self.dict_pixel:
            if i<= seuil:
                other.dict_pixel[0].extend(self.dict_pixel[i])
            else:
                other.dict_pixel[255].extend(self.dict_pixel[i])
        return other

#Réponse question 8
Reg=Region('Paysage')

for i in range(Im.shape[0]):
    for j in range(Im.shape[1]):
        px=Pixel(i,j,Im[i,j])
        Reg.ajouter_pixel(px)

Regbin=Reg.binariser_reg(seuil)
```

PROBLEME 2

Partie I

1.

```
def getData(T) :
    req='select * from ' + T + ';'
    curseur.execute(req)
    l=curseur.fetchall()
    return l
```

2.

```
def rechercheImages(mink , maxk):
    l=getData('image')
    lident=[]
    for e in l:
        taille=3.0*e[3]*e[4]/1024
        if taille<=maxk and taille>=mink :
            lident.append(e[0])
    return lident
```

Partie II

3.

```
SELECT  nomF , hauteur * largeur AS d
FROM  Image
WHERE  extF = 'png'
ORDER BY  d  DESC ;
```

4.

```
SELECT  IdRegion , count (*)
FROM  Pixel
GROUP BY  IdRegion ;
```

5.

```
SELECT  IdImage , count (IdRegion) c
FROM  Image
GROUP BY  IdImage
HAVING  c = 2 ;
```

6.

```
SELECT  Pixel.X , Pixel.Y
FROM  Region , Pixel
WHERE  Region.IdRegion = Pixel.IdRegion AND  Region.label = 'cercle'
AND Pixel.rouge > Pixel.vert  AND  Pixel.rouge > pixel.bleu ;
```

7.

```
UPDATE  Pixel
SET  rouge = 0 , vert = 0 , bleu = 255
WHERE  IdRegion = 'MER102' ;
```

Partie III (5 points)

8.

$$\pi\,(X\,,Y\,)\ \left(\sigma(label\ =\ "Cercle")(Pixel\,)\ \propto\ (Region)\right)_{IdRegion}$$

9.

$$\pi\,(nomF\,)\ \ (Image) - \pi\,(nomF\,)\ \left((Image\,)\ \propto\ (Region)\right)_{IdImage}$$