TD 7: Images en niveau de gris et en noir et blanc

Q1: Définir le tableau à deux dimensions T=[[0,0,0,0],[1,1,1,1],[0.25,0.25,0.75,0.75],[0.25,0.25,0.75,0.75]]

Q2a: Tester les commandes suivantes

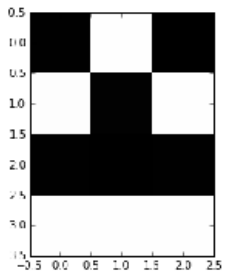
```
import matplotlib.pyplot as plt
plt.imshow(T,cmap='gray',interpolation="none")
```

Vous venez d'afficher l'image en niveau de gris correspondant au tableau T.
Quelle valeur correspond au blanc? au noir? au gris clair?

Q2b: Retester la dernière commande en supprimant interpolation="none". (Dans la suite, la fonction imshow sera utilisée avec l'argument interpolation="none".)

Q3 Tester imshow avec le tableau à deux dimensions à une seule ligne T=[[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]].

Q4: Faire afficher une image correspondant à l'image suivante:



Dans la suite du TP, on appelle image (en niveau de gris) un tableau à deux dimensions dont les élements sont dans $[0, 1]$.

On donne les fonctions suivantes:

```
def ligne(p):
    L=[]
    for i in range(p):
        L.append(0)
    return L
```

et

```
def tableau_noir(n,p):
    T=[]
    for i in range(n):
        L=ligne(p)
        T.append(L)
    return T
```

Q5: Tester ces deux fonctions pour p=3 et n=4. Faire afficher le résultat de **tableau_noir(3,4)** avec imshow.

Dans la suite du TP, lorsqu'une fonction doit renvoyer une image, on utilisera la fonction **tableau_noir** comme fonction auxiliaire afin de créer une première image qu'on modifiera ensuite pour obtenir l'image demandée.

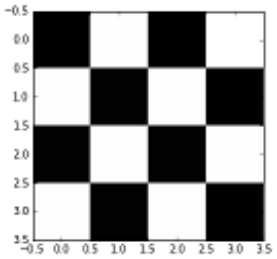
On demande d'écrire des fonctions qui ne provoquent pas d'affichage. On fera systématiquement un affichage des images obtenues afin de tester la correction du programme.

Q6: En vous aidant de l'instruction commentée après la question, écrire une fonction **imageHasard(n)** qui renvoie un tableau T à n lignes et n colonnes dont les valeurs ont été tirées au sort dans $[0, 1]$.

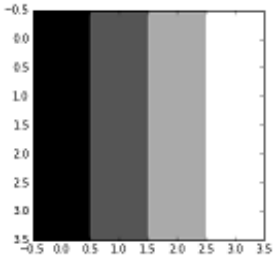
```
import random
x=random.random() # renvoie un un nombre tire au hasard dans [0,1].
```

Q7: La fonction $x \mapsto 1-x$ est un fonction strictement décroissante de $[0,1]$ dans $[0,1]$. Appliquée à un niveau de gris, elle fait passer du noir au blanc et inversement. Ecrire une fonction **négatif(T)** qui renvoie un image N de même taille que T mais correspondant à l'image en négatif de T (on doit donc avoir N[i][j]=1-T[i][j]) Tester cette fonction pour obtenir le négatif d'une image obtenue précédement.

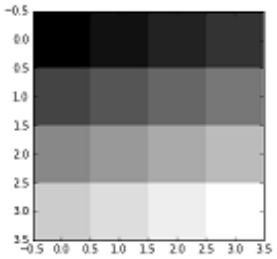
Q8: Dans un damier, une case sur 2 est noire et une case sur 2 est blanche. Les cases (i,j) noires sont celles pour lesquelles i+j est pair. Ecrire une fonction **damier(n)** qui un damier de taille n. (Certains effets indésirables de la pixelisation des images apparaissent pour les grandes valeurs de n).



Q9: On cherche maintenant à obtenir des dégradés de niveau de gris. Pour cela, on fait progressivment passer le niveau de gris du noir au blanc, c'est-à-dire de 0 à 1. Si on veut obtenir p niveau de gris différents, on considèrera un suite arithmétique de niveau de gris dont la première valeur vaut 0 et la $p^{ième}$ vaut 1 donc une suite de raison 1/(p-1).
Ecrire une fonction **degrade1(p)** qui renvoie une image à 1 ligne et p colonnes dont les teintes passent progressivement du noir au blanc lorsqu'on se déplace de droite à gauche (il s'agit de généraliser Q3)
Ecrire une fonction **degrade2(n,p)** qui renvoie une image à n lignes et p colonnes dont les teintes passent progressivement du noir au blanc



Q10: (vous pouvez passer cette question) Ecrire une fonction **degrade_complet(n)** qui renvoie une image à n lignes et n colonnes dont les teintes passent progressivement du noir au blanc comme dans la figure suivante:



Q11: Ecrire une fonction **juxtapose(T1,T2)** ou T1 et T2 sont des images de même nombre de lignes qui renvoie l'image T correspondant à la juxtaposition (horizontale) des images T1 et T2 (T1 à gauche de T2). On remarquera que chaque ligne de T peut être obtenue par concaténation de la ligne de T1 et de la ligne de T2 correspondante.

Q12: Ecrire une fonction **superpose(T1,T2)** ou T1 et T2 sont des tableaux dont le nombre de colonnes est identique et qui renvoie le tableau T correspondant à la superposition des images de T1 et T2 (T1 au dessus de T2).

Q13: L'algorithme suivant permet de tirer un 1 avec une probabilité p:
- Tirer un nombre x au hasard dans $[0, 1]$.
- Si $0 \leq x < p$ affecter 1 à y, sinon affecter 0 à y.

Ecrire une fonction **Bernoulli(p)** qui renvoie 1 avec la probabilité p et 0 avec la probabilité 1-p.

Q14: Ecrire une fonction **noirBlancHasard(n,p)** qui renvoie un tableau T à n lignes et n colonnes dont les valeurs sont des nombres valant 0 ou 1 tirées au sort avec une probabilité de p pour le résultat 1.

Q15: Ecrire une fonction **proportionUns(T)** d'argument un tableau T à deux dimensions (pas forcment égales) dont les élément valent 0 ou 1 et qui renvoie le proportion de 1 dans le tableau T. Tester cette fonction sur un tableau obtenu à l'aide de la fonction précédente.

Nous allons maintenant effectuer des transformations géométriques d'images.

Q16: Ecrire une fonction **inverse(L)** d'argument une liste L qui renvoie la liste des éléments de L dans l'ordre inverse (inverse([2,7,9,1]) doit renvoyer [1,9,7,2].

Q17: Ecrire une fonction **symetrieVerticale(T)** d'argument un tableau T qui renvoie le tableau correspondant à l'image symétrique du tableau T par rapport à son axe vertical.

Q18: Ecrire une fonction **symetrieHorizontale(T)** d'argument un tableau T qui renvoie le tableau correspondant à l'image symétrique du tableau T par rapport à son axe horizontal.

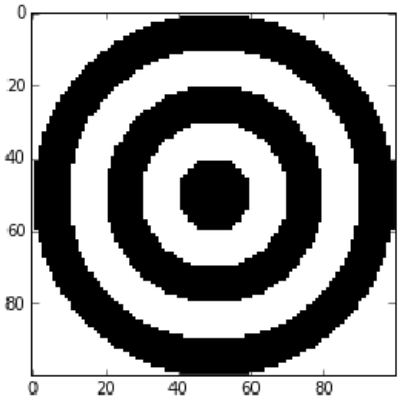
Q19: Ecrire une fonction **quartDeTour(T)** d'argument un tableau T qui renvoie le tableau correspondant à l'image obtenue de T par rotation d'un quart de tour dans le sens horaire.

Monsieur Dupont, professeur au lycée Turing, distribue chaque année la même image à ses élèves. Sa photocopieuse a un défaut de précision: Elle remplace le niveau de gris d'une cellule par la moyenne des niveaux de gris des 9 cellules avoisinantes (moins sur les cotés et les bords). Chaque année, monsieur Dupont garde une nouvelle photocopie pour les photocopies de l'année suivante, la qualité se détériore d'année en année.

Q20: Ecrire une fonction **photocopie(T)** qui simule l'effet décrit précédemment.

Q21: Tester sur une image (par exemple un damier) de votre choix le résultat de 35 années de carrière.
On constate que le test sur le damier n'a pas l'effet escompté mais si l'on examine les valeurs numériques des éléments du tableau obtenu, on constate qu'elles sont très proches. Comment expliquer cette apparente contradiction? Proposer un moyen d'y remédier.

Q22: Ecrire une fonction **cible(n,p)** qui renvoie le tableau à deux dimensions correspondant à une cible comportant p couronnes circulaires concentriques.



Correction:

```
#Q1
T=[[0,0,0,0],[1,1,1,1],[0.25,0.25,0.75,0.75],[0.25,0.25,0.75,0.75]]
#Q2
import matplotlib.pyplot as plt
plt.imshow(T,cmap='gray',interpolation='none')
#Q3
T=[[0,0.2,0.4,0.6,0.8,1]]
plt.imshow(T,cmap='gray',interpolation='none')
#Q4 0 represente le noir et 1 le blanc, lorsque x croit de 0 a 1 le niveau de gris diminue progressivement
#Q5
def tableau_noir(n,p): # fonction initialise_tableau du cours
    T=[]
    for i in range(n):
        l=[] #nouvelle ligne
        for j in range(p):
            l.append(0) #la ligne contient p 0
        T.append(l) #ajout de la nouvelle ligne
    return T

# Q6
import random
def imageHasard(n):
    T=initialiseTableau(n,n)
    for i in range(n):
        for j in range(n):
            T[i][j]=random.random()# tirage au sort dans l'intervalle [0,1]
    return T
#Q7
def negatif(T):
    n,p=len(T),len(T[0])
    N=tableau_noir(n,p)
    for i in range(n):
        for j in range(p):
            N[i][j]=1-T[i][j]
    return N

#Q8
def damier(n):
    T=tableau_noir(n,n)
    for i in range(n):
        for j in range(n):
            if (i+j)%2==1:
                T[i][j]=1
    return T



#Q9a
def degrade1(p):
    T=tableau_noir(1,p)
    for j in range(p): # une seule ligne pas de boucles imbriquees
        T[0][j]=j/(p-1)
    return T

#Q9b
def degrade2(n,p):
```

```
    T=tableau_noir(n,p)
    for i in range(p):
        for j in range(p):
            T[i][j]=j/(p-1)# independant du numero de ligne
    return T

#Q10
def degrade_complet(n,p):
    T=tableau_noir(n,p)
    x=0
    r=1/(n*p-1)
    for i in range(n):
        for j in range(p):
            T[i][j]=x
            x=x+r
    return T

#Q11
def juxtapose(T1,T2): # ligne par ligne
    n1=len(T1)
    T=[]
    for i in range(n1):
        l=T1[i]+T2[i] # concatenation d'une ligne de T1 avec une ligne de T2
        T.append(l)
    return T

# La concatenation evite de manipuler les elements un par un comme dans la fonction suivante:
def juxtapose2(T1,T2): #par elements
    n1,p1,n2,p2=len(T1),len(T1[0]),len(T2),len(T2[0])
    T=tableau_noir(n1,p1+p2)
    for i in range(n1):
        for j in range(p1):
            T[i][j]=T1[i][j]
    for i in range(n2):
        for j in range(p2):
            T[i][j+p1]=T2[i][j]
    return T

#Q11
def superpose(T1,T2): # par tableau
    return T1+T2 # les lignes de T2 viennent apres celles de T1

# on aurait pu faire plus complique:
def superpose1(T1,T2): #par elements
    n1,p1,n2,p2=len(T1),len(T1[0]),len(T2),len(T2[0])
    T=tableau_noir(n1+n2,p1)
    for i in range(n1):
        for j in range(p1):
            T[i][j]=T1[i][j]
    for i in range(n2):
        for j in range(p2):
            T[i+n1][j]=T2[i][j]
    return T

def superpose2(T1,T2): # ligne par ligne
    n1,n2=len(T1),len(T2)
    T=[]
```

```
    for i in range(n1):
        T.append(T1[i])
    for i in range(n2): #les lignes de T2 suivent celles de T1
        T.append(T2[i])
    return T


# Q13
def Bernoulli(p):
    if random.random()<=p: # un nombre tire au hasard dans [0,1] a une
                           # probabilite p d'appartenir a [0,p]
        return 1
    else:
        return 0
# Q14
def noirBlancHasard(n,p):
    T=initialiseTableau(n,n)
    for i in range(n):
        for j in range(n):
            T[i][j]=Bernoulli(p)
    return T

#Q15
def proportionUns(T):
    n,p=len(T),len(T[0])
    compteur=0
    for i in range(n):
        for j in range(p):
            if T[i][j]==1:
                compteur=compteur+1
    return compteur/(n*p) # np est le nombre de cases du tableau
# Q16
def inverse(L):
    I,n=[],len(L)
    for i in range(n):
        I.append(L[n-1-i])
    return I

# Q17
def symetrieVerticale(T):
    n,p=len(T),len(T[0])
    S=[]
    for i in range(n):
        S.append(inverse(L[i])
    return S

# Q18 3 versions differentes
def symetrieHorizontale(T):
    n,p=len(T),len(T[0])
    TS=initialiseTableau(n,p)
    for i in range(n):
        for j in range(p):
            TS[i][j]=T[n-1-i][j]
    return TS

def symetrieHorizontale2(T): # plus simple en inversant l'odre des lignes
    n,p=len(T),len(T[0])
```

```
    TS=[]
    for i in range(n):
        TS.append(T[n-i-1]) # on inverse les lignes de T
    return TS

def symetrieHorizontale3(T): # plus simple en inversant avec la fonction inverse
    return inverse(T)

# Q19
def quartDeTour(T):
    n,p=len(T),len(T[0])
    TR=initialiseTableau(p,n)
    for i in range(p):
        for j in range(n):
            TR[i][j]=T[j][p-1-i] # fair une figure pour comprendre
    return TR


Q20
def moyenne(T,i,j):
    n,p,S=len(T),len(T[0])=0
    if 0<i<n-1:
        if 0<j<p-1:
            for di in range(-1,2):
                for dj in range(-1,2):
                    S=S+T[i+di][j+dj]
            S=S/9
         elif j==0:
            for di in range(-1,2):
                for dj in range(0,2):
                    S=S+T[i+di][j+dj]
            S=S/6
         else: #j=p-1
            for di in range(-1,2):
                for dj in range(-1,1):
                    S=S+T[i+di][j+dj]
            S=S/6
    elif i==0:
        if 0<j<p-1:
            for di in range(0,2):
                for dj in range(-1,2):
                    S=S+T[i+di][j+dj]
            S=S/6
         elif j==0:
            for di in range(0,2):
                for dj in range(0,2):
                    S=S+T[i+di][j+dj]
            S=S/4
         else: #j==p-1
            for di in range(0,2):
                for dj in range(-1,1):
                    S=S+T[i+di][j+dj]
            S=S/4
    elif i==n-1:
        if 0<j<p-1:
            for di in range(-1,1):
                for dj in range(-1,2):
```

```
                    S=S+T[i+di][j+dj]
            S=S/6
         elif j==0:
            for di in range(-1,1):
                for dj in range(0,2):
                    S=S+T[i+di][j+dj]
            S=S/4
         else: #j=p-1
            for di in range(-1,1):
                for dj in range(-1,1):
                    S=S+T[i+di][j+dj]
            S=S/4
    return S
def photocopie(T):
    n,p=len(T),len(T[0])
    P=tableau_noir(n,p)
    for i in range(n):
        for j in range(n):
            P[i][j]=moyenne(T,i,j)
    return P
Q21
T21=damier(10)
for i in range(35):
    T21=photocopie(T21)
plt.imshow(T21,cmap='gray',interpolation='none')
```

La représentation en niveau de gris attribue en fait le noir au niveau minimum présent dans le tableau et le blanc au niveau maximum de manière à renforcer le contraste de l'image. Pour éviter ce renforcement, on peut remplacer le niveau d'un pixel arbitraire par 0 et le niveau d'un pixel arbtraire par 1 avant l'affichage.

```
Q22
def cible(n,p):
    T=tableau_noir(n,n)
    R=n/2
    for i in range(n):
        for j in range(n):
            d=((i-R)**2+(j-R)**2)**0.5
            if d>R:
                T[i][j]=1
            else:
                x=int(p*d/R)%2
                T[i][j]=x
    return T
```