## Exercice 1 : Négatif

Pour obtenir le négatif d'une image, toutes les composantes $x$ de tous les pixels de l'image sont remplacées par $255 - x$.
Obtenir le négatif de la photo « tours.JPG ».

```
from PIL import Image
import numpy as np

im =Image.open("tours.JPG")
tab=np.array(im, dtype="uint8")
print(tab)
"""négatif"""
tab_new = 255 -tab
photo=Image.fromarray(tab_new)
photo.show()
```

## Exercice 2 : filtre rouge

Chaque pixel de l'image est une combinaison de rouge, de vert et de bleu. En assignant la valeur 0 aux composantes vertes et bleue, on obtient l'image de droite.
Obtenir le filtre rouge de la photo « tours.JPG ».

```
from PIL import Image
import numpy as np

im =Image.open("tours.JPG")
tab=np.array(im)
print(tab)

"""filtre rouge"""
tab_new = np.copy(tab)
tab_new[:,:,1:3]=0
photo=Image.fromarray(tab_new)
photo.show()
```

## Exercice 3 : niveau de gris

Dans une image en niveaux de gris, chaque pixel est noir, blanc, ou à un niveau de gris entre les deux. Cela signifie que les trois composantes ont la même valeur. L'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités de couleurs fondamentales, mais une moyenne pondérée. La formule standard donnant le niveau de gris en fonction des trois composantes est :

$$gris = (0.299 \cdot rouge + 0.587 \cdot vert + 0.114 \cdot bleu)$$

Obtenir la photo « tours.JPG » en niveaux de gris.

```
from PIL import Image
import numpy as np

im =Image.open("tours.JPG")
tab=np.array(im)
print(tab)

"""niveau de gris"""
#gris = int(round(0.299·rouge + 0.587·vert + 0.114·bleu))
l,c,p=np.shape(tab)
tab_new=np.zeros((l,c),dtype="float")
tab_new[:,:]=((0.299*tab[:,:,0])+np.round(0.587*tab[:,:,1])
+(0.114*tab[:,:,2]))
tab_new=np.array(tab_new,dtype="uint8")
photo=Image.fromarray(tab_new)
photo.show()
```

## Exercice 4 : Seuillage

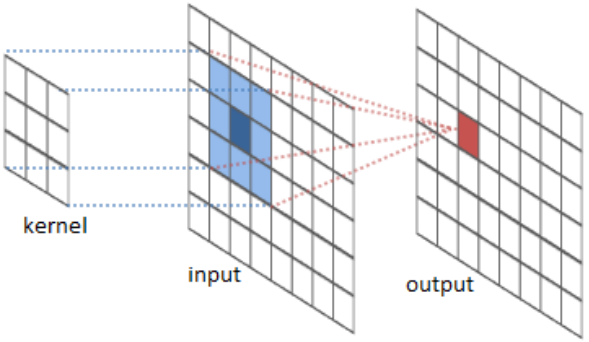
Le seuillage d'image est la méthode la plus simple de segmentation d'image. À partir d'une image en niveau de gris, le seuillage d'image peut être utilisé pour créer une image comportant uniquement deux valeurs, noir ou blanc (monochrome). On remplace un à un les pixels d'une image par rapport à une valeur seuil fixée (par exemple 123). Ainsi, si un pixel à une valeur supérieure au seuil, il prendra la valeur 255 (blanc), et si sa valeur est inférieure, il prendra la valeur 0 (noir).

En utilisant un mask (tableau de booléen aux dimensions de la photo), obtenir le seuillage de la photo « tours.JPG

```
"""seuillage version gris"""
l,c,p=np.shape(tab)#dimension du tableau
tab_new=np.zeros((l,c),dtype="uint8")
tab_new[:,:]=0.299*tab[:,:,0]+0.587*tab[:,:,1]+0.114*tab[:,
:,2]
mask=tab_new[:,:]>123
tab_new[mask]=255
tab_new[np.invert(mask)]=0
photo=Image.fromarray(tab_new)
photo.show()
"""seuillage vesrions couleur"""
mask1=tab[:,:,0]>123
mask2=tab[:,:,1]>123
mask3=tab[:,:,2]>123
mask=mask1+mask2+mask3
tab_new=np.copy(tab)
tab_new[mask]=255
tab_new[np.invert(mask)]=0
photo=Image.fromarray(tab_new)
photo.show()
```

## Exercice 5 : Détection de contours suivant une direction

On peut effectuer une transformation mathématique (appelé produit de convolution) permettant de modifier la valeur d'un pixel en fonction des valeurs des pixels avoisinants, affectées de coefficients.




Pour simplifier nous travaillerons avec une photo en niveau de gris. Le filtre est représenté par un tableau (une matrice appelée kernel), caractérisé par ses dimensions et ses coefficients, dont le centre correspond au pixel concerné. Le tableau ci-dessous permet de détecter les contours dans la direction horizontale.

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

$$tab[i,j] = abs(-tab[i-1,j-1] + tab[i-1,j+1] - 2tab[i,j-1] + 2tab[i,j+1] - tab[i+1,j-1] + tab[i+1,j+1])$$

En utilisant une photo en niveau de gris, des tableaux numpy de type float, appliquer le filtre moyenneur ci-dessus à l'image « tours.JPG »

```
"""détection_contour"""
l,c=np.shape(tab)
tab_new=np.zeros((l,c),dtype="float")
tab=np.array(tab,dtype="float")#nécessaire !!!!!
tab_new[1:-1,1:-1]=abs(-1*tab[:-2,:-2]+0*tab[:-2,1:-
1]+1*tab[:-2,2:]
-2*tab[1:-1,:-2]+0*tab[1:-1,1:-1]+2*tab[1:-1,2:]
-1*tab[2:,:-2]+0*tab[2:,1:-1]+1*tab[2:,2:])

tab_new=np.array(tab_new,dtype="uint8")
photo=Image.fromarray(tab_new)
photo.save("photo_contour_H.jpg") # pour l'enregistrer au
format voulu
photo.show()
```

## Exercice 6 : Symétrie axial

On note $l$ et $c$ le nombre de lignes et de colonnes de la photo « tours.JPG ». On envisage une symétrie axiale avec un axe vertical. Il convient donc d'échanger chaque pixel d'indice $[i,j]$ par le pixel d'indice $[i,c-1-j]$ avec $i$ allant de 0 à $l-1$ et $j$ allant de 0 à la partie entière de $c-1$.

1) Écrire la fonction *sym_V*("*image*") qui renvoie la version de l'image ayant subi cette opération
2) Écrire de même une fonction *sym_H*("*image*") réalisant une symétrie axiale avec un axe horizontal.

```
"""question 1"""
def sym_ver(image):
    nom=image+".JPG"
    tab=Image.open(nom)
    tab=np.array(tab,dtype="uint8")
    tab_new=np.copy(tab)
    l,c,p=np.shape(tab)
    for i in range(l):
        for j in range(c):
            tab_new[i,j]=tab[i,c-1-j]
            tab_new[i,c-1-j]=tab[i,j]
    photo=Image.fromarray(tab)
    photo.show()
    photo=Image.fromarray(tab_new)
    photo.show()

"""question 2"""
def sym_H(image):
    nom=image+".JPG"
    tab=Image.open(nom)
    tab=np.array(tab,dtype="uint8")
    tab_new=np.copy(tab)
    l,c,p=np.shape(tab)
    for i in range(l):
        for j in range(c):
            tab_new[i,j]=tab[l-1-i,j]
            tab_new[l-1-i,j]=tab[i,j]
    photo=Image.fromarray(tab)
    photo.show()
    photo=Image.fromarray(tab_new)
    photo.show()
```

## Exercice 7 : Réduction et agrandissement

On dispose d'une image de taille $[l, c]$ et on souhaite la réduire. Nous supposons que la réduction consiste à diviser la longueur et la largeur par un nombre entier $d$. C'est le cas le plus simple et on envisage de garder une ligne sur $d$ lignes et une colonne sur $d$ colonnes.

1. Écrire une fonction $reduction("image", d)$ qui renvoie la version de l'image ayant subi cette opération, avec $d$ entier.

Pour l'agrandissement d'un facteur d entier, le procédé simple consiste à copier d fois chaque colonne puis d fois chaque ligne.

2. En utilisant la fonction précédente, récupérer le tableau *tab* correspondant à une réduction de 10 de l'image « tours.JPG » puis écrire une fonction *agrandissement* qui renvoie la version de l'image ayant subie cette opération un agrandissement de 10.

```
"""question 1"""
def reduction(image,d):
    nom=image+".JPG"
    tab=Image.open(nom)
    tab=np.array(tab,dtype="uint8")
    l,c,p=np.shape(tab)
    tab_new=np.zeros((l//d,c//d,p))
    tab_new=np.array(tab_new,dtype="uint8")
    l,c,p=np.shape(tab)
    for i in range(l//d):
        for j in range(c//d):
            tab_new[i,j]=tab[i*d,j*d]
    photo=Image.fromarray(tab)
    photo.show()
    photo=Image.fromarray(tab_new)
    photo.show()
    return tab_new

tab=reduction("tours",10)

"""question 2"""

def agrandissement(tab,d):
    l,c,p=np.shape(tab)
    tab_new=np.zeros((l*d,c*d,p))
    tab_new=np.array(tab_new,dtype="uint8")
    l,c,p=np.shape(tab)
    for i in range(l):
        for j in range(c):
            tab_new[i*d:i*d+d,j*d:j*d+d]=tab[i,j]
    photo=Image.fromarray(tab)
    photo.show()
    photo=Image.fromarray(tab_new)
    photo.show()
```

## Exercice 8 : Rotation à 90°

La suite se limite donc à des images carrées ($l = c = n$) avec une rotation d'un quart de tour. Une rotation d'un quart de tour consiste à permuter les pixels des quatre quadrants. On note $p_1$ le pixel d'indice $[i, j]$, $p_2$ le pixel d'indice $[n - 1 - j, i]$, $p_3$ le pixel d'indice $[n - 1 - i, n - 1 - j]$, $p_4$ le pixel d'indice $[j, n - 1 - i]$. On effectue pour $i$ allant de 0 à n/2 et $j$ allant de 0 à n/2 les permutations $(p_1, p_2, p_3, p_4) \rightarrow (p_2, p_3, p_4, p_1)$

A l'aide deux boucles for, écrire une fonction rotation(image) appliquant cette méthode pour renvoyer une version de l'image ayant subie cette rotation

```
def rotation(image):
    nom=image+".JPG"
    tab=Image.open(nom)
    tab=np.array(tab,dtype="uint8")
    tab_new=np.copy(tab)
    l,c,p=np.shape(tab)
    print(l,c)
    n=l
    for i in range(n//2):
        for j in range(n//2):
            tab_new[i,j]=tab[n-1-j,i]
            tab_new[n-1-j,i]=tab[n-1-i,n-1-j]
            tab_new[n-1-i,n-1-j]=tab[j,n-1-i]
            tab_new[j,n-1-i]=tab[i,j]
    photo=Image.fromarray(tab)
    photo.show()
    photo=Image.fromarray(tab_new)
    photo.show()

rotation("tours_carrees")
```

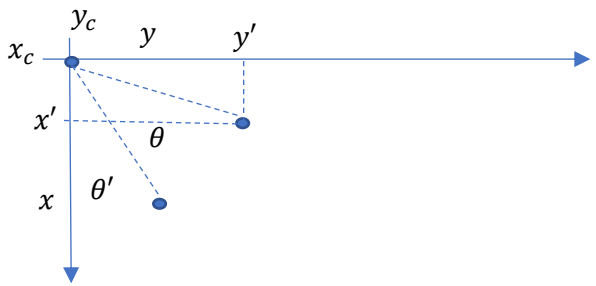
## Exercice 9 : Rotation quelconque

Un tableau de pixel associé à une photo en niveau de gris peut être vu comme une fonction discrète $u(x_i, y_j)$ qui associé une valeur de codée en uint8. Si l'on souhaite effectuer une rotation des valeurs de ce tableau alors cela revient à créer une nouvelle fonction $v$ telle que :

$$v(x_i, y_j) = u\left(T(x_i), T'(y_j)\right)$$

Les fonctions $T$ et $T'$ décrivent la rotation imposée.

1) Montrer qu'une rotation de $\theta$ conduit aux changements de coordonnées suivants :




$$x' = T(x) = x_c + ((x - x_c)cos\theta + (y - y_c)\, sin\theta)$$

$$y' = T'(y) = y_c + (-(x - x_c)sin\theta + (y - y_c)\, cos\theta))$$

On donne ci-dessous la fonction permettant de transformer les valeurs continues de $x'$ et $y'$ en valeurs discrètes entières respectant les dimensions initiales de la photo.

```
def interpole(u,X,Y):
    ny,nx=np.shape(u)
    X_approx=np.array(np.minimum(nx-1,np.maximum(0,np.floor(X))),dtype="int")
    Y_approx=np.array(np.minimum(ny-1,np.maximum(0,np.floor(Y))),dtype="int")
    return u[X_approx,Y_approx]
```

2) Ecrire une fonction prenant pour argument un tableau de photo en niveau de gris et permettant une rotation d'un angle $\theta$ de la photo.

La nouvelle du pixel position après rotation est :

$$sin\theta' = \frac{y - y_c}{R}$$

$$cos\theta' = \frac{x - x_c}{R}$$

$$\sin(\theta + \theta') = cos\theta' sin\theta + sin\theta' cos\theta = \frac{y' - y_c}{R}$$

$$\cos(\theta + \theta') = cos\theta' cos\theta - sin\theta' sin\theta = \frac{x' - x_c}{R}$$

$$x' = T(x) = x_c + ((x - x_c)cos\theta + (y - y_c)\, sin\theta)$$

$$y' = T'(y) = y_c + (-(x - x_c)sin\theta + (y - y_c)\, cos\theta))$$

```
def rotation(u,theta,xc,yc):
    nx,ny=np.shape(u)
    y,x=np.meshgrid(np.arange(ny),np.arange(nx))
    X=xc+((x-xc)*np.cos(theta*np.pi/180)+(y-
yc)*np.sin(theta*np.pi/180))
    Y=yc+((y-yc)*np.cos(theta*np.pi/180)-(x-
xc)*np.sin(theta*np.pi/180))
    return interpole(u,X,Y)

#photo=Image.fromarray(tab)
#photo.show()
tab_new=rotation(tab_new,45,l//2,l//2)
photo=Image.fromarray(tab_new)
photo.show()
```

Exercice 10 : Kernels

Pour réaliser des opérations sur les images, on peut utiliser des filtres à matrices de convolution. Ceux-ci consistent à modifier la valeur d'un pixel en fonction des valeurs des pixels voisins.

Considérons une image en niveau de gris et regardons un pixel et ses voisins. On souhaite modifier le pixel central en appliquant une pondération comme représenté ci-dessous :

| | | |
|---|---|---|
| $p_1$ | $p_2$ | $p_3$ |
| $p_4$ | $p_5$ | $p_6$ |
| $p_7$ | $p_8$ | $p_9$ |

**Tab. 1** – Valeurs des pixels (niveau de gris)

| | | |
|---|---|---|
| $c_1$ | $c_2$ | $c_3$ |
| $c_4$ | $c_5$ | $c_6$ |
| $c_7$ | $c_8$ | $c_9$ |

**Tab. 2** – Pondération des pixels

La valeur du pixel central est remplacé comme suit :

$$p_5 = \frac{\sum_{i=1}^{9} c_i p_i}{\sum_{i=1}^{9} c_i} \quad \text{si} \quad \sum_{i=1}^{9} c_i \neq 0$$

et

$$p_5 = \sum_{i=1}^{9} c_i p_i + 128 \quad \text{si} \quad \sum_{i=1}^{9} c_i = 0\,.$$

Si la valeur est plus grande que 255 ou plus petite que 0, on écrêtera simplement la valeur à 255 ou à 0.

Réaliser un programme python filtre(image,matrice) prenant en entrée une image en niveau de gris et une matrice 3 × 3 donnant les coefficients de la pondération du filtrage. Pour simplifier, on supprimera dans l'image renvoyée les lignes et colonnes de contour sur lequel le filtre ne s'applique pas.

On pourra tester le programme avec les filtres suivants :

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

**Tab. 3** – Rien

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

**Tab. 4** – Floutage

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

**Tab. 5** – Contours

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

**Tab. 6** – Contraste 1

| | | |
|---|---|---|
| -1/6 | -2/3 | -1/6 |
| -2/3 | 13/3 | -2/3 |
| -1/6 | -2/3 | -1/6 |

**Tab. 7** – Contraste 2

| | | |
|---|---|---|
| -2 | -1 | 0 |
| -1 | 0 | 1 |
| 0 | 1 | 2 |

**Tab. 8** – Relief

```
def filtre(tab,kernel):
    l,c=np.shape(tab)
    tab_new=np.zeros((l,c))
    s=np.sum(kernel)
    if s!=0:
        for i in range(1,l-1):
            for j in range(1,c-1):
                tab_new[i,j]=(kernel[0,0]*tab[i-1,j-
1]+kernel[0,1]*tab[i-1,j]+kernel[0,2]*tab[i-1,j+1] +\
                kernel[1,0]*tab[i,j-
1]+kernel[1,1]*tab[i,j]+kernel[1,2]*tab[i,j+1] +\
                kernel[2,0]*tab[i+1,j-
1]+kernel[2,1]*tab[i+1,j]+kernel[2,2]*tab[i+1,j+1])/s
                if tab_new[i,j]>255:
                    tab_new[i,j]=255
                if tab_new[i,j]<0:
                    tab_new[i,j]=0
    else:
        for i in range(1,l-1):
            for j in range(1,c-1):
                tab_new[i,j]=(kernel[0,0]*tab[i-1,j-
1]+kernel[0,1]*tab[i-1,j]+kernel[0,2]*tab[i-1,j+1] +\
                kernel[1,0]*tab[i,j-
1]+kernel[1,1]*tab[i,j]+kernel[1,2]*tab[i,j+1] +\
                kernel[2,0]*tab[i+1,j-
1]+kernel[2,1]*tab[i+1,j]+kernel[2,2]*tab[i+1,j+1])+128
                if tab_new[i,j]>255:
                    tab_new[i,j]=255
                if tab_new[i,j]<0:
                    tab_new[i,j]=0
    tab_new=np.array(tab_new[1:-1,1:-1],dtype="uint8")
    return tab_new
```