# Travaux Pratiques n°9
# Traitement des Images

▷ Récupérer les fichiers du TP9 sur `moodle Déodat`.

Les décompresser puis les placer dans un nouveau répertoire `TP9` de votre PC. Y ajouter votre photo (format .png) et vérifier que les fichiers `Plage.png` et `Plage.bmp` sont bien présents.

▷ Ouvrir `Python`.

▷ **Exercice 1 : Manipulations simples.**

▷ Charger la photo `Plage.png` sous forme de matrice (appelée `Photo`) puis l'afficher à l'aide de Python à l'aide du script ci-dessous. Pensez à vous placer dans le bon répertoire à l'aide du module `os`.

```
# Import des modules :
from matplotlib.pyplot import *
import matplotlib.image as mpimg
from numpy import *

# Import de l'image :
Photo=mpimg.imread('Plage.png')

# Affichage :
imshow(Photo)
show()
```

Une image est codée par une matrice à trois dimensions : le nombre de lignes, de colonnes, et la profondeur, de longueur 3. La matrice est un tableau numpy. La profondeur de chaque case contient trois réels codant une couleur sous le format RGB (rouge, vert, bleu). Le poids pour chaque couleur est compris entre 0 et 1 ((0,0,0) pour le noir, (1,1,1) pour le blanc, format .png) ou c'est un entier entre 0 et 255 (format .bmp).
On accède à l'élément d'indices i, j et k avec la syntaxe Photo[i,j,k]. On peut également utiliser la technique du slicing vue dans les précédents TPs.

▷ Récupérer la taille de cette matrice (`Photo.shape`). On notera $m$ et $n$ le nombre de lignes et de colonnes : `m,n=Photo.shape[:2]`.

▷ Tester et comprendre l'instruction suivante :

```
imshow(Photo[600:,:1000,:])
```

▷ Tester et comprendre les instructions suivantes :

```
Photo2=zeros([m,n,3])
for k in range(3):
    Photo2[:,:,k]=(Photo[:,:,0]+Photo[:,:,1]+Photo[:,:,2])/3
imshow(Photo2)
```

▷ Tester et comprendre les instructions suivantes :

```
for i in range(m):
    for j in range(n):
        Photo[i,j,1]=0
        Photo[i,j,2]=0

imshow(Photo)
```

▷ **Exercice 2 : Superposition de deux images.**

Le but de cet exercice est d'incorporer votre portrait à l'image `Plage.png`.

Il faudra pour ceci supprimer le décor, c'est-à-dire le tableau de la classe.

▷ Récupérer votre photo sous `Python`, l'afficher, stocker sa taille $(m_2, n_2)$.

▷ Créer une fonction `tableau` qui reçoit un triplet de flottants $(r, g, b)$ et décide (en renvoyant `True` ou `False`) si ce triplet désigne la couleur du tableau. On admet, bien que ce soit perfectible, que le triplet $(r, g, b)$ doit être supprimé si les conditions suivantes sont vérifiées :

$$0{,}11 \leqslant r \leqslant 0{,}80 \qquad 0{,}20 \leqslant g \leqslant 0{,}85 \qquad 0{,}13 \leqslant b \leqslant 0{,}80$$
$$-0{,}03 \leqslant g - b \leqslant 0{,}14 \qquad -0{,}09 \leqslant b - r \leqslant 0{,}10 \qquad -0{,}17 \leqslant r - g \leqslant 0{,}01$$

On pourra tester cette fonction en remplaçant les pixels représentant le tableau sur le portrait par des pixels noirs.

En jouant sur les conditions on peut améliorer le résultat.

▷ Remplacer les pixels de la matrice `Photo` de la plage par les pixels de votre portrait, à l'endroit de votre choix, en omettant ceux qui représentent le tableau.

Afficher le résultat.

▷ **Exercice 3 : Symétrie, rotation et détection de contours.**

Ici, nous allons chercher à obtenir un effet miroir, une rotation et la détection des contours de la photo `Plage.png`.

▷ Proposer un algorithme permettant de réaliser un effet miroir et le tester. On pourra compléter l'algorithme suivant :

```
def symetrie(matB):
    nb_lig,nb_col,nb_coul=matB.shape
    matC=np.zeros((nb_lig,nb_col,nb_coul))
    for i in range(nb_lig):
```

```
5           for j in range(nb_col):
6               for k in range(nb_coul):
7                   ??????????
8       return matC
```

▷ Sur le même modèle, proposer un algorithme permettant d'effectuer une rotation de $\pi/2$ de la photo `Plage.png`. Afficher le résultat.

La reconnaissance de formes dans une image est une composante importante de l'analyse d'images. Elle se décompose en plusieurs étapes qui consistent à extraire les contours des objets dans l'image afin de les reconnaitre ou d'en détecter le mouvement. La première de ces étapes est la mise en évidence des contours des objets dans l'image. C'est cette étape que nous allons aborder très succinctement.

Un contour définit la limite d'un objet dans une image. Cette limite est caractérisée par un changement dans l'image : un changement de couleur ou de contraste. Ce changement se traduit dans la valeur des pixels qui sont localisés de part et d'autre de la limite. Nous sommes donc à la recherche d'un moyen de détecter et de localiser un changement.

Considérons un pixel p(i,j) dans une image couleur (liste ou type array avec les 3 valeurs R, G, B). Il s'agit ensuite de mesurer la différence, une "distance", entre notre pixel de référence et ses voisins (On pose $p_1 = p(i-2,j)$, $p_2 = p(i,j-2)$, $p_3 = p(i+2,j)$, $p_4 = p(i,j+2)$) en utilisant une fonction de norme standard. On se propose d'utiliser la fonction suivante : $norme = \sqrt{(distance(p_1,p_3))^2 + (distance(p_2,p_4))^2}$ On donne la fonction distance :

```
1 def distance(L1,L2):
2     d=0
3     for i in range(len(L1)):
4         d+=(L1[i]-L2[i])**2
5     return sqrt(d)
```

Si cette distance est "grande", c'est-à-dire au-delà d'un certain seuil (que l'on fixera à 1), cela signifie que le pixel est sur un contour, on l'affiche en noir.

Si cette distance est "petite", c'est-à-dire en deça du seuil, cela signifie qu'il n'y a pas de rupture de couleur. Le pixel n'est donc pas sur un contour, on l'affiche en blanc.

▷ Créer une fonction `DetectionContour` qui prend en argument la photo dont on souhaite délimiter le contour et qui renvoie une photo faisant apparaître seulement les contours. Tester sur l'image `Plage.png`. Faire varier la valeur de seuil et observer les différences.

▷ **Exercice 4 : Floutage d'une image.**

Dans cette partie, nous nous intéressons à la mise en place d'un "floutage" d'une image. Pour réaliser un floutage par moyenne simple sur la matrice de pixels, il faut lui appliquer un filtre, que l'on appelle également un masque. Afin de bien comprendre ce principe, nous proposons d'étudier un exemple de filtrage. On considère les matrices :

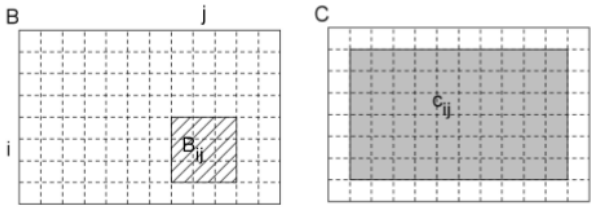
$$A = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

et

$$B = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 \\ -5 & -6 & -7 & -8 & -9 & -10 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 3 & 3 & 4 & 4 \\ 0 & 0 & 1 & 3 & 3 & 3 \end{pmatrix}.$$

Pour chaque élément *bij* de B, on considère la matrice 3*3 *Bij* qui l'entoure, on calcule le produit de convolution de A par *Bij* **(multiplication coefficient par coefficient)** et on note *cij* la somme des coefficients de la matrice produit obtenue (on pourra utiliser la fonction *np.sum* sur une liste).

Si *bij* est un élément en bordure de B, on posera *cij* = *bij*. On forme ainsi une nouvelle matrice C dont les éléments intérieurs sont les *cij* (et les éléments au bord sont les *bij*).




On dit qu'on a filtré la matrice B par la matrice A, ou qu'on a appliqué le masque (filtre) A sur l'image B.

▷ Créer la fonction `filtrer1(filtreA,matB)` qui prend en argument une matrice carrée *filtreA* de dimension *taille*taille* (taille est un entier impair = 3) et une matrice quelconque *matB* de dimensions supérieures, et qui renvoie la matrice C. On remarquera que si taille>3, la bordure devra être plus épaisse.

On souhaite maintenant appliquer le filtre A à une matrice de pixels B, c'est à dire aux 3 tableaux B[ :, :,0], B[ :, :,1], B[ :, :,2] et enregistrer le résultat dans une matrice C de même format que B.

▷ Créer une fonction `filtrer(filtreA,matB)` qui prend en argument une matrice carrée *filtreA* de dimension *taille*taille* et un tableau numpy *matB* de dimensions $n \times p \times 3$,et qui renvoie le tableau C de dimensions identiques. Vous pourrez calculer successivement C[ :, :,0], C[ :, :,1], C[ :, :,2] à l'aide d'une boucle. Tester sur l'image `Plage.bmp`. Le format .bmp ne donne pas accès à la valeur des pixels entre 0 et 1 (flottants) mais en entiers entre 0 et 255.

Ces matrices réalisent un floutage par moyenne simple (coefficients tous égaux, de somme 1). Plus la taille du filtre est grande, plus le flou sera fort.