TP 5 - Traitement d'image matricielle (bitmap)

I. Image matricielle

➜ Il existe principalement deux catégories d'images :

➢ les images bitmap, ou images pixellisées : ensemble de points (pixels) dans un tableau. Exemple de formats/fichiers : `.bmp, .jpeg, .png, .gif, .tiff`
Chaque point se caractérise par une ou plusieurs valeurs numériques définissant sa couleur.

- Chaque point de couleur est une liste de trois nombres codant le rouge, le vert et le bleu (RGB). Ces nombres peuvent être entiers ($[\![0;255]\!]$), soit réel ($[0;1]$) : ceci est défini dans le type de fichier.
- Si les trois valeurs sont égalent, cela donne du gris : le noir est codé `[0,0,0]`.
- Les fichier `.png` accepte un quatrième nombre pour la transparence : 0 pour la transparence totale et 1 pour l'image pleine.

➢ les images vectorielles : représentation d'entités géométriques définies par une formules mathématiques. Un cercle peut être défini par deux points désignant un diamètre ; pour un carré on peut considérer deux points désignant une diagonale ; une courbe sera définie par des points et une équation ...
C'est le processeur qui traduira ces formes en informations interprétables par la carte graphique.

Voici quelques fonctions du module `matplotlib.pyplot` :

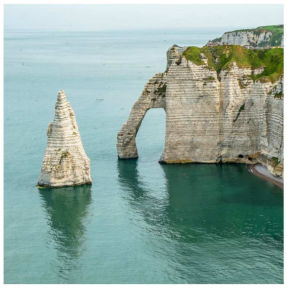
Notation	Explication
`f=imread('fichier')`	Transforme une image en un tableau de nombres
`imsave('fichier',f)`	Transforme un tableau nombre en image
`imshow(f)`	Visualise l'image associée au tableau `f`

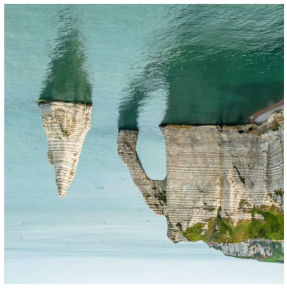
Attention ! Les fichiers `.png` sont codés sur $[0;1]^3$.

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> f=plt.imread('Etretat.png') # pour lire
>>> plt.imshow(f)  # pour visualiser
>>> np.shape(f)   # pour la taille
(500, 500, 4)
>>> f[0,0]
array([ 0.266, 0.498, 0.756, 1.], dtype=float32)
```

Image retournée horizontalement

```
>>> h=f[::-1]
>>> plt.imsave('Etretat2.png',h) # pour enregistrer
```





Lecture et sauvegarde des fichiers

Il est important d'avoir conscience qu'il y a un **répertoire de travail** (celui du `shell`) qui peut différer de celui du fichier contenant le script. Il convient de donner à Pyzo la position des fichier manipuler (en écriture ou en lecture).

⇛ Méthode 1 : Sous Pyzo, on peut redémarrer le `Shells` dans le répertoire d'un script (fichier `.py` enregistré) :
- Menu `Exécuter/Démarrer le script` : `Ctrl+Mal+E`

⇛ Méthode 2 : Utiliser les fonctions du module os

Notation	Explication
`import os`	Importe le module os
`os.getcwd()`	Donne le répertoire courant
`os.chdir('I:\Python')`	Change le répertoire courant
`os.mkdir(TP')`	Créer un répertoire `TP` dans le répertoire courant

⇛ Méthode 3 : Décrire l'adresse exacte du fichier à chaque appel (lecture ou écriture)

```
f=plt.imread('I:\Python\TP\Etretat.png')
plt.imsave('I:\Python\TP\Etretat2.png',f)
```

Mise en route

➜ **Pour bien commencer la séance :**
- récupérer le dossier dans le répertoire commun de la classe en faisant un copier-coller sur votre clé usb (ou dans `Mes documents`)
- Depuis Pyzo, ouvrir le fichier `intro.py`.
- Appliquer la méthode 1 (ci-dessus) pour exécuter le fichier `intro.py` afin que la répertoire de travail du `Shells` devienne le même que celui où le script et la photo sont enregistrés :
  menu `Exécuter/Démarrer le script`
- Vérifier que la photo `Etretat.png` est affichée et que le fichier `Etretat2.png` a été créé dans le même répertoire.

Exercices de traitements d'une image

Exercice 1 Vous pouvez traiter cet exercice sans retaper les scripts, uniquement par lecture :

1. Que donnent les instructions suivantes ?

```
f=np.ones((100,100,3))
plt.imshow(f)
```

?

```
g=1*f  # copie de matrice
g[:,:,1]=np.zeros((100,100))
plt.imshow(g)
```

?

2. Que fait la fonction suivante :

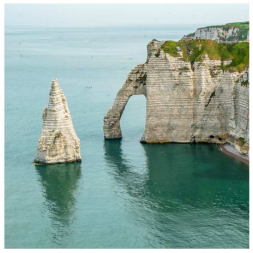
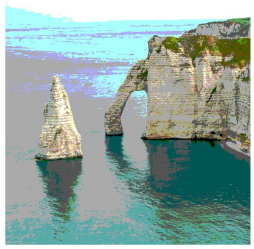
```
def transfo(image):
 f=plt.imread(image)
 t=np.shape(f)
 for i in range(t[0]):
  for j in range(t[1]):
      f[i,j,:3]=np.array([f[i,j][2],f[i,j][0]
                                  ,f[i,j][1]])
 plt.imsave('T_'+image,f)
 plt.imshow(f)
 plt.show()
```

?

Exercice 2 Compléter la fonction suivante qui retourne une image (de format .png) en niveau de gris. On prendra comme teinte de gris : 30% de rouge, 59% de vert et 11% de bleu.
La variable f_in est le nom de l'image initiale ; la variable f_out est le nom de l'image transformée.

```
1  def gris(f_in,f_out):
2      f=plt.imread(f_in)
3      l,c,d=np.shape(f)
4      g=np.zeros((...,...,...))
5      for ...:
6          for ...:
7              for ...:
8                  g[i,j,k]=...
9      plt.imshow(g)
10     plt.show()
11     plt.imsave(f_out,g)
```

Exercice 3 Écrire une fonction `posterise(f_in,f_out,n)` qui redéfinie les teintes de chaque pixel sur n nuances ; c'est-à-dire que les valeurs des teintes, par défaut dans $[0,1]$, sont répartis sur n valeurs fixées, par exemple $\left\{\frac{k}{n-1}; k \in [\![0, n-1]\!]\right\}$.

Exercice 4 *Éclaircir* Définir une fonction `eclaircir(f_in,f_out)` qui retourne l'image éclaircie tout en perdant le moins possible de contraste ; que la valeurs des teintes doivent être modifier pour éclaircir l'image tout en assurant que deux teintes différentes au départ, demeurent différentes après transformation.

Exercice 5 *Flouter une image* Écrire une fonction `flouter(f_in,f_out,d)` qui permet de flouter une image : pour chaque pixel, la teinte devient la moyenne des teintes des pixels qui l'entourent.
Le paramètre `d` donne le nombre de pixels à considérer tout autour. Par exemple, pour `d=2`, une teinte sera la moyenne de 25 teintes, un carrée de côté `2d+1`.
Pour simplifier, on ne floutera pas les bords de l'image (une bande de largeur `d-1`).

Exercice 6 *Redimensionner une image* Écrire un script pour redimensionner une image, à proportion conservée : `redim(ligne,f_in,f_out)`.
Le paramètre `ligne` est la taille souhaitée pour la ligne de la nouvelle image.

On pourra, éventuellement, discuter suivant si c'est une réduction ou un agrandissement.