L'essentiel en une 1/2-page

```python
# Bibliothèques nécessaires dans ce TP
import numpy as np
import matplotlib.pyplot as plt

# Manipulation des tableaux numpy

t=np.array([1,2]) # matrice ligne (1 2)
3*t # renvoie (3 6)
t+t # renvoie (2 4)

t=np.array([[1,2],[3,4]]) # matrice carrée de taille 2

# Copier un tableau
s=t.copy() # s est un tableau similaire à t

# Tableau nul
np.zeros((3,2)) # renvoie la matrice nulle de M_{3,2}(R).

# Ouverture et affichage d'une image au format png
I=plt.imread("chemin/image.png")
# I est tableau numpy.ndarray (un tableau de tableaux)
# ndarrayarray = vecteur multi-dimensionnel.
plt.imshow(I)
plt.show()

# dimensions du tableau I
n,p,q=I.shape # Tableau de n lignes, p colonnes.
# La valeur de q est fixée à q=3 dans ce TP.
# On verra en effet qu'un pixel correspond à un triplet.
```

On travaillera avec le fichier `plage.png` disponible sur le réseau.

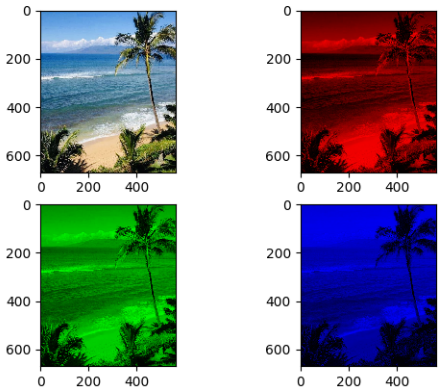
1. Commencez par ouvrir et faire afficher cette image par Python.
2. Affichez les dimensions de cette image.

On manipulera donc cette image comme une matrice à $n = 670$ lignes, chaque ligne étant composée de $p = 564$ colonnes et chaque élément de cette matrice est un triplet ($q = 3$).

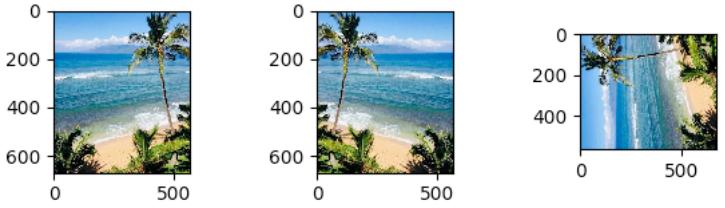
Exercice 1: (Solution)

Une image en couleur est représentée par une matrice de pixels. Un pixel est un triplet de valeurs comprises entre 0 et 1. Chaque composante de ce triplet est associé à une couleur primaire : Rouge, Vert, Bleu. On parle de code RGB.
Un pixel noir sera codé par $(0, 0, 0)$ et un pixel blanc par $(1, 1, 1)$.
Si les trois composantes ont la même valeur, cela code une certaine nuance de gris. Lorsqu'une seule composante est non nulle, l'image prend alors une teinte de la couleur primaire correspondante.

1. Écrire les fonction `rouge(image)`, `vert(image)`, `bleu(image)` d'argument un tableau correspondant à une image et renvoyant un tableau correspondant à la même image où l'on a effacé le nuances des deux autres couleurs primaires.



2. Écrire une fonction `symétrie(image)` renvoyant la transformée par symétrie verticale et une fonction `rotation(image)` renvoyant la rotation d'angle $+\frac{\pi}{2}$.



Exercice 2: (Solution)

Pour convertir une image en couleur en une image en noir et blanc, il faut convertir chaque pixel en niveau de gris. Pour cela, on associe à chacune des trois composantes, la même valeur appelée luminance, qui représente la luminosité du pixel. On utilisera la définition suivante de la luminance d'un pixel (R, G, B) :

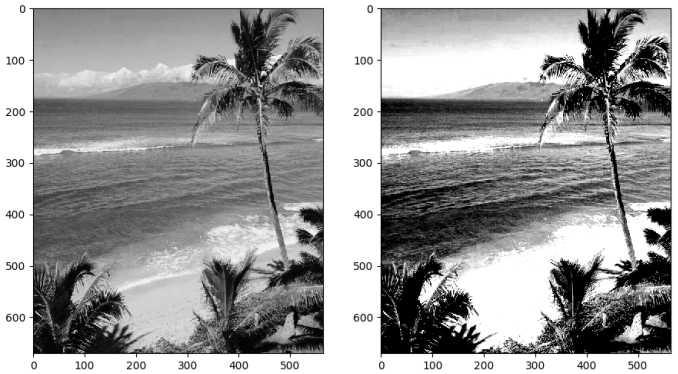
$$x = 0,2126 \times R + 0,7152 \times G + 0,0722 \times B.$$

1. Écrire une fonction `gris(image)` qui renvoie l'image en noir et blanc.
2. Pour augmenter le contraste d'une image, on applique une transformation affine sur la luminance :

$$f(x) = a \times (x - 0,5) + 0,5.$$

La contraste augmente, si $a > 1$. Attention, on veillera à remplacer une valeur négative par 0 et une valeur plus grande que 1 par 1.

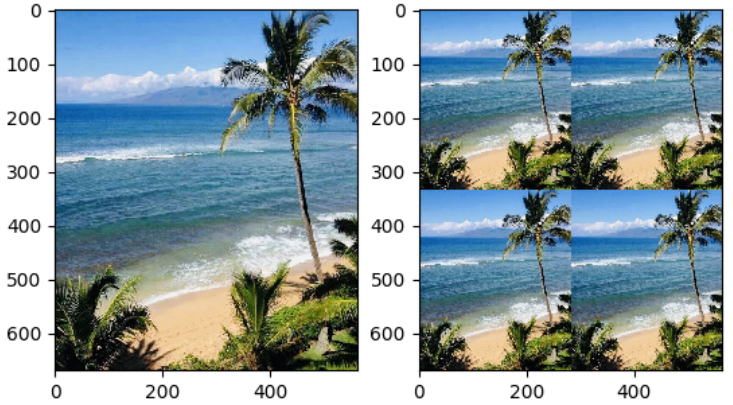
Écire une fonction `contrate(image,a)` renvoyant une image contrastée suivant le coefficient a. L'image entrée en argument est en niveau de gris.



Exercice 3: (Solution)

On souhaite découper l'image en 4 petites photos d'identité comme dans un photomaton. Pour cela, on considère les pixels par blocs de 4 (2×2) pour lesquels les quatre pixels se retrouveront dans une des 4 images différentes. Pour effectuer cette transformation, les dimensions de l'image doivent être paires. Aucun pixel ne doit disparaitre au cours de la transformation, ils sont simplement déplacés.

1. Écrire une fonction `photomaton(image)` effectuant cette transformation.



2. Avec le fichier `mona.png` (disponible sur le réseau), effectuer et afficher 10 itérations du processus de photomaton.

Exercice 4: (Solution)

Pour cette partie, on utilisera les codes RGB codés sur $24 = 3 \times 8$ bits : chaque composante R, G et B est donc codée sur un octet et a une valeur entière comprise entre 0 et 255.
On peut convertir un tableau de flottants en transformant les valeurs en entiers comme suit :

```
1 image_24bits=(255*image[:,:,:3]).astype(np.uint8)
```

La stéganographie consiste à cacher un message dans un autre. Il est ainsi possible de cacher une image dans une autre de la manière suivante : en considérant que chaque couleur primaire d'un pixel est codée sur un octet, il est possible d'arrondir ces valeurs au multiple de 16 le plus proche. Ainsi, le code couleur ne nécessite que 4 bits (ceux de poids fort) sur 8 huit bits disponibles. On utilise alors les 4 bits restants pour y cacher une image.

1. Écrire une fonction `retrouve(image)` qui prend une image stéganographiée et renvoie l'image cachée.
   Faire le test avec `steg.png`
2. Écrire une fonction `steg(image1,image2)` qui cache l'image 2 dans l'image 1.

Exercice 5: (Solution)

On veut recolorer une image en diminuant le nombre de couleurs. Pour cela, on utilise le principe itératif suivant :

- On crée une liste `L` de `N` couleurs choisies aléatoirement, `N` étant le nombre de couleurs que l'on souhaite obtenir.
- On répète le processus suivant avec un nombre prédéfini d'itérations :
  - Pour chaque pixel de l'image `I`, on détermine la couleur de `L` dont ce pixel est le plus proche au sens de la distance euclidienne, en utilisant les trois coordonnées du code `RGB`.
  - On crée une liste `NL` de taille `N`, telle que `NL[i]` est la couleur qui est le barycentre (c'est-à-dire moyenne composante par composante) des pixels dont `L[i]` est le plus proche.
  - On recolore chaque pixel de `I` par la couleur `NL[i]` si `L[i]` est le plus proche de ce pixel.
  - On remplace `L` par `NL`.

1. Écrire une fonction `couleur_proche(pixel,L)` d'arguments un tableau de taille `3` correspondant à une couleur ainsi qu'une liste `L` de couleurs et qui renvoie l'indice `i` tel que `L[i]` est la couleur la plus proche de `pixel`
2. Écrire une fonction `iteration(I,L)` qui calcule une itération du processus décrit précédemment et renvoie le couple `NI,NL` correspondant à l'image modifiée et la nouvelle liste de couleurs.
3. En déduire une fonction `recoloration(im,N,k)` qui prend en argument une image `im` et un entier `N` correspondant au nombre de couleurs autorisées et renvoie l'image modifée selon l'algorithme décrit précédemment, en effectuant k itérations. On pourra utiliser la commande suivante pour créer une liste aléatoire de couleurs :

```
1 L=[np.random.rand(3) for i in range(N)]
```

Remarques

Le temps de calcul peut être long, jusqu'à une minute par itération.