Informatique en CPGE (2014-2015)
Devoir maison 3

L'objectif est d'écrire un programme qui copie une image en couleur au format bmp 24 bits, puis qui lui applique des transformations.

Le fichier programme et les images doivent être stockés dans un même dossier.

Il est conseillé de revoir le chapitre 7 (Les fichiers) du cours de première année.

Partie 1

Dans le format bmp couleur 24 bits, le fichier est constitué d'un en-tête de 54 octets donnant les caractéristiques du fichier, puis du codage des pixels (3 octets = 24 bits par pixel). Dans l'en-tête, après les 18 premiers octets, 4 octets donnent le nombre de colonnes puis 4 octets donnent le nombre de lignes.

1. Récupérer quelques images de tailles variables au format bmp couleur 24 bits (ou dans un autre format et les convertir dans ce cas en bmp 24 bits).
2. Le programme demande à l'utilisateur le nom du fichier (par exemple "image.bmp") qu'il stocke dans une variable "nom".
3. Ensuite le programme ouvre le fichier en lecture mode binaire (**f=open(nom,'rb')**) et lit les 18 premiers octets qu'il stocke dans une variable "debut_entete" (**debut_entete = f.read(18)**).
4. Le programme lit et stocke les 4 octets suivant dans une variable "colonnes", puis encore les 4 octets suivant dans une variable "lignes".
5. Si les 4 octets donnant le nombre de colonnes sont $o_1o_2o_3o_4$, alors le nombre de colonnes s'obtient par : **nbcolonnes** $= o_1 + 256o_2 + 256^2o_3 + 256^3o_4 = o_1 + 256(o_2 + 256(o_3 + 256o_4))$. Il en est de même pour les lignes. Le programme affiche alors les valeurs des variables "nbcolonnes" et "nblignes".
6. Ne pas oublier de fermer le fichier (**f.close()**).
7. Vérifier avec des images de tailles différentes que le nombre de colonnes et de lignes affichés dans la console est correct.

Partie 2

1. Le programme lit maintenant les 28 octets restant de l'en-tête stockés dans une variable "fin_entete".
2. Il ouvre un fichier "nouveau.bmp" en écriture mode binaire (**g=open('nouveau.bmp','wb')**) et y écrit le début de l'en-tête (**g.write(debut_entete)**), les octets représentant le nombre de colonnes, les octetes représentant le nombre de lignes, et la fin de l'en-tête.
3. A partir du nombre de colonnes, déterminer le nombre d'octets utilisés pour chaque ligne de l'image. Attention, dans la norme bmp il faut un multiple de 4 octets par lignes et donc on rajoute éventuellement des octets nuls à la fin d'une ligne. Par exemple si le nombre de colonnes est 155, il faut $155 \times 3 = 465$ octets pour les pixels de la ligne et il nécessaire de rajouter 3 octets pour obtenir un multiple de 4. Créer une variable entière "nboctetnuls" qui représente ce nombre d'octets nuls et le calculer en fonction du nombre de colonnes (on utilise le reste d'une division euclidienne).
4. Le programme va lire chaque ligne du fichier d'origine, pixel par pixel et l'écrire dans le fichier nouveau.bmp. Pour cela on utilise une boucle sur le nombre de colonnes imbriquée dans une boucle sur le nombre de lignes, on lit un pixel (**pixel=f.read(3)**) et on l'écrit dans le nouveau fichier. Lorsqu'on a lu et écrit tous les pixels de la ligne, on lit et on écrit les éventuels octets nuls de fin de ligne.
5. Fermer les deux fichiers qui ont été ouverts en lecture et en écriture. Faire fonctionner le programme et vérifier que l'on obtient une copie exacte de l'image.

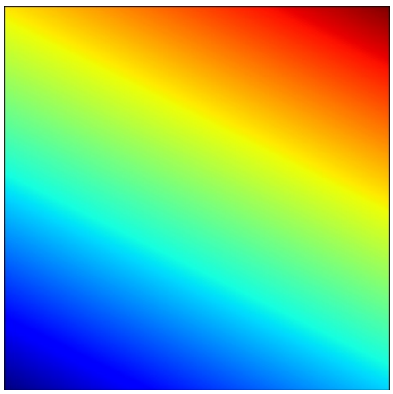
Partie 3

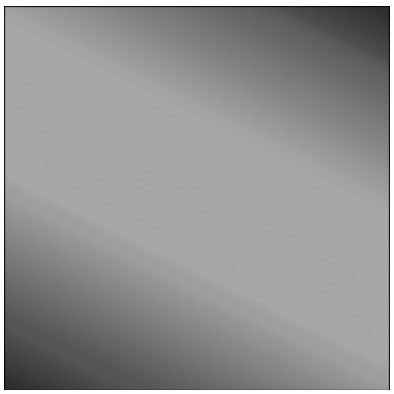
Maintenant que l'on sait recopier un fichier image pixel par pixel, on peut lui faire subir des transformations.

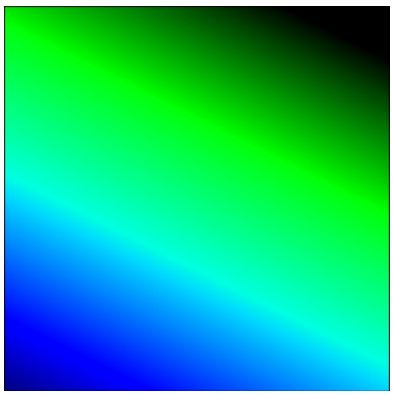
1. Pour obtenir une image en niveaux de gris, on va calculer pour chaque pixel une moyenne des trois composantes rouge, vert, bleue. Donc après la lecture d'un pixel, on calcule la moyenne de pixel[0], pixel[1] et pixel[2], (attention cette variable doit avoir une valeur entière). On crée ensuite le nouveau pixel (**pixelnew=bytes(3*[moyenne])**) et on écrit ce nouveau pixel.
2. On peut supprimer la composante rouge, ou la verte, ou la bleue :
   par exemple **pixelnew=bytes([pixel[0],0,pixel[2]])**.
3. On peut inverser la quantité de chaque couleur avec l'instruction :
   pixelnew=bytes([255-pixel[0],255-pixel[1],255-pixel[2]]).
4. Compléter le programme afin qu'il demande à l'utilisateur la transformation à appliquer en début de programme, et affiche à la fin l'image transformée.

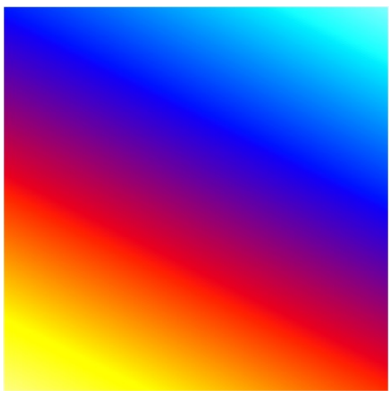
   Par exemple sous Windows avec le logiciel Paint :
   pour lancer Paint directement, on écrit au début du fichier la commande :
   ```
   from os import system
   ```
   puis à la fin du fichier, on écrit :
   ```
   system("C:/windows/system32/mspaint.exe nouveau.bmp")
   ```
   (adresse de l'exécutable mspaint.exe et nom du fichier transformé).

Exemples de résultats :









Partie 4

Pour les transformations ci-dessus, l'écriture de la nouvelle image se fait "à la volée", on lit un pixel et on l'écrit. Pour d'autres transformations il est nécessaire de stocker les pixels ; pour cela on utilise une matrice.

1. Créer une matrice **matrice** remplie de valeurs **None**. Le nombre de lignes est **nbLignes** et le nombre de colonnes est **nbColonnes** plus éventuellement 1 si **nbOctetsNuls** est différent de zéro.
2. On sépare maintenant la lecture de l'image et son écriture. A la lecture, chaque pixel et éventuellement **octetNuls** sont stockés dans **matrice[i][j]**.
3. Programmer simplement l'écriture afin de retourner verticalement l'image initiale.
4. Programmer simplement l'écriture afin de retourner horizontalement l'image initiale.
5. Imaginer d'autres transformations.

Exemples :