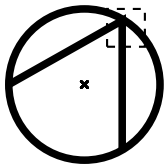
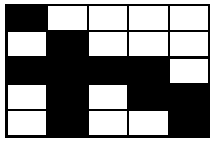
Par la suite, nous allons découvrir des moyens de coder des images.

Prenons l'image suivante, elle est composée d'un cercle de rayon 1 centré sur le point O qui est marqué d'une croix. On y voit deux cordes jointes reliant les points d'affixes $e^{i\pi}$ à $e^{i\frac{\pi}{3}}$ puis à $e^{-i\frac{\pi}{3}}$.
Si on zoome alors on peut avoir un rendu pixellisé comme ci-contre.





Pour résumer, on a vu deux manières de coder une même image :
◇ de manière ..... : la forme est décrite (expl : tracer un vecteur $\overrightarrow{v}$ rouge à partir d'un point A, un cercle noir de centre O et de rayon R, etc. Cet encodage permet un zoom de l'image sans perte de détails.

expl : .....

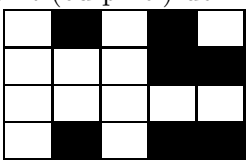
◇ de manière ..... (ou ..... (carte de points)) : l'image est considérée comme un grand tableau de points de couleurs (penser à une mosaïque). Chaque point de la matrice s'appelle un pixel (**pi**cture **el**ement). Lors d'un zoom, la qualité de l'image se détèriore malheureusement. La taille de l'image est souvent très volumineuse.

expl : ..... (formats non compressés) ou ..... (formats compressés)

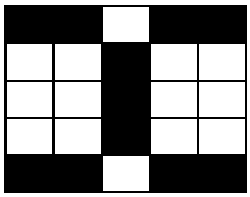
Par la suite, on ne se place que dans le codage de manière matricielle.

I Le noir et blanc

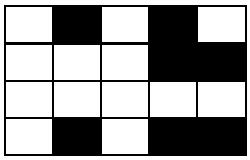
Chaque point (ou pixel) de l'image ne peut avoir que deux couleurs : le noir (..) ou le blanc (..).

expl : l'image  est codée par $\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$

Cette image (comme sa matrice) comporte ... lignes et ... colonnes. Puisque chaque case est codée par 1 bit, il faut un total de ..... bits.

Exo 1 : Encoder dans une matrice l'image suivante : 

Pour gagner de la place mémoire, on peut proposer une compréhension relativement simple à mettre en œuvre : la compression RLE (Run-Length Encoding - codage par plages). Elle consiste à donner la taille d'une plage de pixels contiguës et changer de couleur à chaque changement de plage. On commence par la couleur par défaut (ici noire).

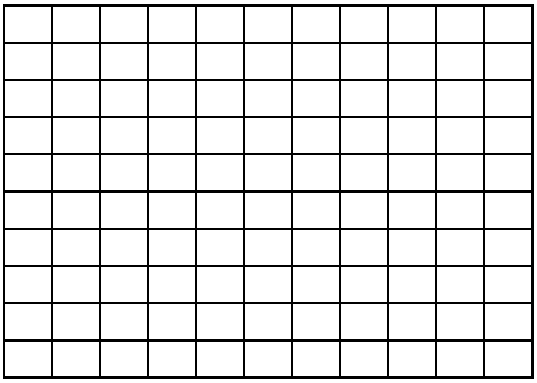
expl : l'image  est codée par ligne par

0 1 1 1 1 1
0 3 2
0 5
0 1 1 1 2

ou parfois par 01111426112

Exo 2 : recréer l'image compressée par

0 2 1 5 1
0 3 1 3 1
0 2 7
0 2 1 1 3 1 1
0 1 2 1 3 1 2
11
1 1 7 1 1
1 1 1 5 1 1 1
0 1 3 3 3
0 4 3



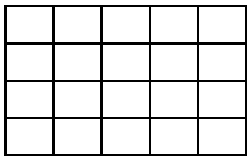
On remarquera que ce type de compression n'est pas vraiment efficace lorsque .....
.....

II 256 nuances de gris

Plutôt que coder la couleur à l'aide de deux états, on va utiliser 256 niveaux de gris. Du noir codé par .. au blanc codé par ..... Les différents gris seront codés selon leur teinte plus ou moins foncée (plus noir le gris sera, plus son nombre sera petit).

On code ce gris à l'aide d'un octet (8 bits) car $2^8 = 256$.

1 Manipulation & implémentation

On considère la matrice $M1 = \begin{pmatrix} 0 & 0 & 255 & 255 & 255 \\ 255 & 0 & 255 & 255 & 255 \\ 255 & 0 & 0 & 0 & 0 \\ 255 & 255 & 0 & 0 & 255 \end{pmatrix}$, dessiner l'image codée : 

Pour accèder à la valeur d'un élément de M (une case), on entre $M[n^o\, ligne][n^o\, colonne]$ (en partant de 0).

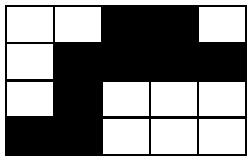
Préciser les valeurs suivantes : $M[0][0] = \ldots$; $M[0][2] = \ldots$; $M[2][0] = \ldots$; $M[2][3] = \ldots$; $M[3][4] = \ldots$

Afin de parcourir toutes les cases de la matrice et afficher la valeur de l'élément à la ligne i et la colonne j (*i.e.* $M[i][j]$), on propose l'implémentation suivante. Une fois la matrice parcourue, le fichier 'tmp.bmp' est créé et contient l'image codée par la matrice.

```
# definition de la matrice M
M = numpy.array([[0,255,255,255],[0,0,255,255],[0,0,255,0]])
for i in range(len(M)):              # donne le nombre de ligne de M
    for j in range(len(M[i])):       # donne le nombre de colonnes de la ligne i
        print(M[i][j])
Nom2 = 'test.bmp'
CreerImgGris(M, CheminImage+Nom2) # creation du fichier tmp.bmp
```

Exo 3 :

1. Télécharger puis dézipper dans votre répertoire le zip [chap08-codageImage.zip].
Ouvrir le fichier `ISN-chap08-codageImage-eleve.py`, changer la variable `CheminImage` afin d'atteindre votre répertoire puis charger les premières lignes du code. Quelle est la taille affichée de l'image `champigris.bmp` ?
2. Écrire les lignes de code `python` ci-dessus et vérifier l'image obtenue.
3. Modifier la définition de M afin de coder l'image définie par la matrice M.
4. On souhaite retourner l'image selon un axe horizontal, *i.e.* obtenir l'image ci-contre :
Où la case (1,1) est-elle envoyée ? la case (2,1) ? (3,1) ? (2,1) ? (2,2) ?
5. Proposer les coordonnées de la nouvelle case en fonction des coordonnées de l'ancienne case (i,j).
6. Proposer un algorithme puis l'implémenter sous python.



III La couleur avec 256 nuances

Sur les écrans cathodiques ou LED, chaque pixel est constitué de la somme de trois couleurs (Rouge, Vert, Bleu) dont on règle l'intensité lumineuse. Le mélange de ces couleurs forme ainsi l'ensemble du spectre lumineux.

Le codage RGB (.....) est un système de codage dans lequel chaque composante de couleur est codée à l'aide de 256 nuances (1 octet). Ainsi un pixel est codé à l'aide de ... octet souvent représenté en hexadécimal (codage en base 16).

expl : $\overline{A}^{16} = 10$; $\overline{B}^{16} = 11$; $\overline{C}^{16} = \ldots$; $\overline{F}^{16} = \ldots$;
$\overline{10}^{16} = \ldots$; $\overline{7A}^{16} = \ldots$; $\overline{B0}^{16} = \ldots$; $\overline{FF}^{16} = \ldots$

expl : (128,0,255)=#8000FF.

Voici quelques exemples de couleurs (*cf.* [http://rapidtables.com/web/color/RGB_Color.htm])

(0,0,255) correspond à du ..... (128,0,255) correspond à du .....

(0,0,0) correspond à du ..... (255,255,0) correspond à du .....

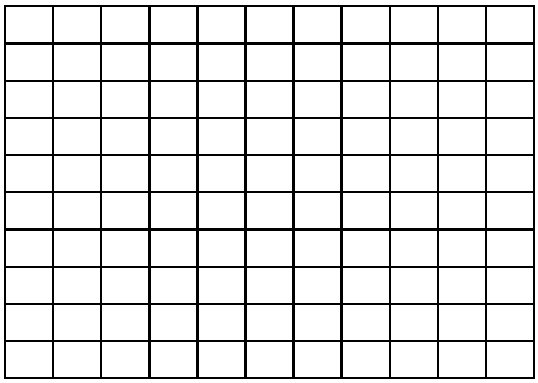
Ce système nécessitant beaucoup de place mémoire, les fichiers enregistrés sous format GIF réduit la palette de couleurs à 256. Ainsi seuls 8 octets (au lieu de 24) sont nécessaires au codage.

Exo 4 : Compléter le code de la fonction `composanteBleue()` et observer le résultat une fois le code chargé.

Travail à faire pour la prochaine séance

Exo 5 :

Trouver l'image compressée par
0319111713157211151145114131121
1111111112111111111211111111121



Exo 6 :

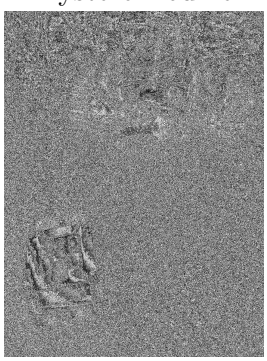
1. Écrire un code qui permet de calculer le niveau de gris moyen de l'image (il faudra parcourir toute la matrice et ...)
2. Écrire un code qui permet d'augmenter le contraste de l'image (si le pixel possède un niveau de gris inférieur à 127 alors la valeur deviendra 0, sinon 255).
3. Transformer votre code en deux fonctions `grisMoyen(matrix)` et `contraste(matrix)`
4. Appliquer le code à l'image `'champigris.bmp'` et sauvegarder dans `'champiNB.bmp'`.

Exo 7 :

Des apprentis espions ont essayé de coder leur images afin qu'elles deviennent difficilement compréhensibles.
Pour la première, l'espion1 a décalé de -400 pixels horizontalement et verticalement l'image.
Pour la deuxième, le codage consiste à remplacer le niveau de gris de chaque pixel (noté ng) par le reste de la division du produit $73 \times ng$ par 256.
Pour annuler ce codage, il faut trouver le nombre a tel que le reste de la division du produit $a \times 73$ par 256 fasse 1, puis faire la même manipulation.

mystereDecale

mystereBrouille



Ainsi à partir de la matrice M, l'espion2 a remplacé chaque case `M[i,j]` par `(M[i,j]*37)%256`. À partir de la matrice mystère MM, il vous faut donc remplacer chaque case par `(MM[i,j]*a)%256`.

Sauriez-vous retrouver les images initiales à partir des images mystères ? (nb : les deux images mystères se trouvent dans [mysteres.zip]).

Exo 8 :

1. Proposer un code `python` qui permette d'obtenir l'image `'champiNB.bmp'` retournée selon un axe vertical.
2. Proposer un code `python` qui permette d'obtenir l'image `'champiNB.bmp'` retournée selon la diagonale $y = x$.
3. (a) En partant de l'image par la matrice $M1$, la dessiner en lui faisant faire une rotation de $+\frac{\pi}{2}$.
   (b) Où la case (1,1) est-elle envoyée ? La case (2,1) ? La case (3,1) ? La case (2,1) ? La case (2,2) ?
   (c) Proposer les coordonnées de la nouvelle case en fonction des coordonnées de l'ancienne case (i,j).
   (d) Proposer un code `python` qui permette d'obtenir l'image `'champigris.bmp'` tournée à $+\frac{\pi}{2}$.