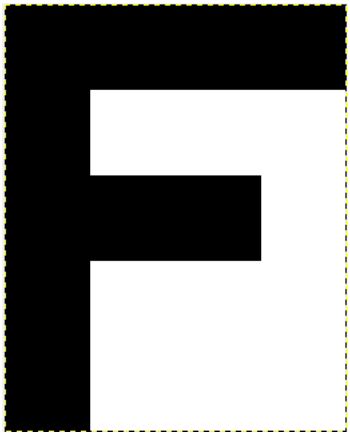
Fichiers textes et images bitmap

Exercice 1 *Images binaires au format PBM*

Le format PBM est un format de fichier texte permettant de stocker des images bitmap en noir et blanc. La première ligne du fichier précise le format avec l'identifiant P1, la seconde définit la largeur L et la hauteur H de l'image et à partir de la troisième ligne le tableau de pixels est stocké ligne par ligne, chaque pixel étant codé par 0 pour un pixel blanc et 1 pour un noir.
On donne ci-dessous l'exemple d'une image de définition 4×5 représentant un F.

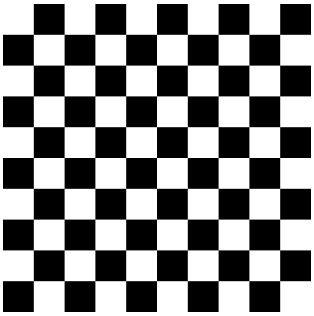
```
P1
4 5
1 1 1 1
1 0 0 0
1 1 1 0
1 0 0 0
1 0 0 0
```



1. Le pixel blanc situé à droite de la barre centrale du F est en ligne d'index 2 et colonne d'index 3.

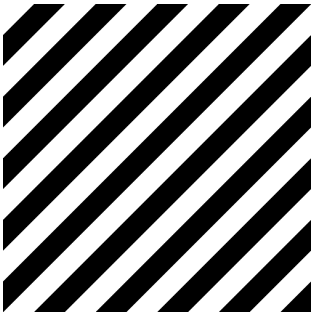
2. Le code ci-dessous permet de créer l'image d'un damier de définition 500×500 dont les cases sont des carrés de 50 pixels. La suite de caractères `'\n'` code un caractère spécial, le **saut de ligne**.

```
L, H = 500, 500
f = open('damier.pbm', 'w')
f.write('P1\n')
f.write(str(L) + ' ' + str(H) + '\n')
for lig in range(H):
    for col in range(L):
        f.write(str((lig // 50 + col // 50) % 2) + ' ')
    f.write('\n')
f.close()
```



Le code ci-dessous génère une image, de définition 500×500, constituée de bandes noires parallèles dont les côtés coincidant avec un bord de l'image, mesurent 50 pixels.

```
L, H = 500, 500
f = open('bandes.pbm', 'w')
f.write('P1\n')
f.write(str(L) + ' ' + str(H) + '\n')
for lig in range(H):
    for col in range(L):
        f.write(str(((lig + col) // 50) % 2 ) + ' ')
    f.write('\n')
f.close()
```



Une version plus complexe du code serait celle-ci :

```
L, H = 500, 500
f = open('bande.pbm', 'w')
f.write('P1\n')
f.write(str(L) + ' ' + str(H) + '\n')
for lig in range(H):
    for col in range(L):
        if lig % 50 + col % 50 >= 50:
            f.write(str((1 + lig // 50 + col // 50) % 2 ) + ' ')
        else:
            f.write(str((lig // 50 + col // 50) % 2 ) + ' ')
    f.write('\n')
f.close()
```

Elle fait apparaître la différence entre `(lig + col) // 50` et `lig // 50 + col // 50` qui est la seule expression différente entre les codes générant un damier ou des bandes.

Si on pose les divisions euclidiennes de `lig` et `col` par 50, on obtient `lig` $= q \times 50 + r$ avec $0 \leqslant r < 50$ et `col` $= q' \times 50 + r'$ avec $0 \leqslant r' < 50$. Dans le code, r est représenté par `lig % 50` et r' par `col % 50`.

Par définition de l'opérateur //, on a lig // 50 $= q$ et col // 50 $= q'$ et donc lig // 50 + col // 50 $= q + q'$.

D'autre part, on a (lig + col) // 50 $= q + q' + (r + r')//50$ et puisque $0 \leqslant r + r' < 100$ on a $(r + r')//50 = 0$ si $0 \leqslant r + r' < 50$ et $(r + r')//50 = 1$ sinon.

3.

```
def inverser_couleurs_pbm(source, but):
    """Lit le fichier pbm source et le recopie dans le fichier pgm
    but en inversant les couleurs"""
    #ouverture de fichiers en lecture pour source et écriture pour but
    f = open(source, 'r')
    g = open(but, 'w')
    #on recopie l'en-tête (les deux premières lignes)
    for k in range(2):
        lig = f.readline()
        g.write(lig)
    #pour les lignes codant les pixels on inverse chaque valeur de pixel
    for lig in f:
        for caractere in lig:
            if caractere == '0':
                g.write('1')
            elif caractere == '1':
                g.write('0')
            else:
                g.write(caractere)
    g.close()
    f.close()
```

Exercice 2 *Images en niveaux de gris au format PGM*

1.

```
def rechercher_pixel_ppm(source, lig, col):
    """Retourne la valeur du pixel en ligne lig et colonne col
    dans le fichier image source au format PGM"""
    f = open(source, 'r')
    for k in range(3 + lig + 1):
        f.readline()
    liste_pixels = f.readline().split()
    f.close()
    return int(liste_pixels[col])
```

2.

```
def flip_ppm(source, but):
    """Retourne un fichier but au format PGM obtenu en appliquant une
        symétrie
    d'axe vertical (flip) à l'image du fichier source au format PGM"""
    #ouverture de fichiers en lecture pour source et écriture pour but
    f = open(source, 'r')
    g = open(but, 'w')
    #on recopie l'en-tête (les trois premières lignes)
    for k in range(2):
        lig = f.readline()
        g.write(lig)
    for lig in f:
        liste_pixels = lig.split()
        g.write(' '.join(liste_pixels[::-1]) + '\n')
    g.close()
    f.close()
```

3.

```
def flop_ppm(source, but):
    """Retourne un fichier but au format PGM obtenu en appliquant une
        symétrie
    d'axe horizontal (flop) à l'image du fichier source au format PGM"""
    #ouverture de fichiers en lecture pour source et écriture pour but
    f = open(source, 'r')
    g = open(but, 'w')
    #on recopie l'en-tête (les trois premières lignes)
    for k in range(3):
        lig = f.readline()
        g.write(lig)
    liste_lignes = f.readlines()
    for lig in reversed(liste_lignes):
        g.write(lig)
    g.close()
    f.close()
```

Fichiers binaires et images bitmap

Exercice 3 *Transformation du photomaton*

1.

```
def transformation(i, j, n):
    """Retourne les coordonnées (colonne, ligne) de l'image du pixel de
    coordonnées (i, j) par la transformation du photomaton"""
    if i % 2 == 0:
        if j % 2 == 0:
            return (i // 2, j // 2)
        else:
            return (i // 2, (j + n) // 2)
    else:
        if j % 2 == 0:
            return ((i + n) // 2, j // 2)
        else:
            return ((i + n) // 2, (j + n) // 2)
```

Puisque `n` est pair dans le contexte de cet exercice on a pour tout entier `i` compris entre 0 et `n` l'égalité entre `(j + n) // 2` et Puisque `n` est pair dans le contexte de cet exercice on a pour tout entier `i` compris entre 0 et `n` l'égalité entre `j // 2 + n // 2` et on peut écrire une version sans structure conditionnelle :

```
def transformation2(i, j, n):
    """Retourne les coordonnées (colonne, ligne) de l'image du pixel de
    coordonnées (i, j) par la transformation du photomaton"""
    parite_x = i % 2
    parite_y = j % 2
    quotient_x = i // 2
    quotient_y = j // 2
    return (quotient_x + parite_x*(n//2), quotient_y + parite_y*(n//2))
```

2.

```
def copie(tableau):
    """Retourne la copie profonde d'un tableau à deux dimensions"""
    return numpy.array([[tableau[j][i] for i in range(len(tableau[j]))]
    for j in range(len(tableau))], dtype = 'uint8')


def photomaton(tableau):
    """Retourne le tableau calculé obtenu après application de la
        transformation
    du photomaton au tableau passé en paramètre"""
    tableau2 = copie(tableau)
    n = len(tableau)
    for i in range(n):
        for j in range(n):
            (i2, j2) = transformation(i, j, n)
            tableau2[i2][j2] = tableau[i][j]
    return tableau2
```

3.
```
def photomaton_iterer(source, k):
    im = Image.open(source)
    nom, extension = source.split('.')
    tab = numpy.asarray(im)
    for iteration in range(1, k + 1):
        tab = photomaton(tab)
        im = Image.fromarray(tab)
        im.save(nom + '-photomaton-iteration-' + str(iteration) + '.' +
            extension)
```

4. Si on itère 9 fois la fonction photomaton à partir de l'image `lenagray.png`, de dimensions 512×512, on retrouve l'image initiale.

5. Si on itère 8 fois la fonction photomaton à partir de l'image `lenagray-256.png`, de dimensions 256×256, on retrouve l'image initiale.

Si on itère 7 fois la fonction photomaton à partir de l'image `lenagray-128.png`, de dimensions 128×128, on retrouve l'image initiale.

On peut conjecturer que n itérations successives de la fonction photomaton sur une image de dimensions $2^n \times 2^n$, permettent de retrouver l'image initiale.

Pour démontrer cette propriété, on peut raisonner sur les écritures en base 2 des coordonnées (colonne, ligne) = $(i,\ j)$ de chaque pixel de l'image source de dimensions $2^n \times 2^n$. On peut démontrer que chaque application de la transformation du photomaton sur le même pixel revient à faire une permutation circulaire sur l'écriture binaire sur n bits de chaque coordonnée. Après n itérations, l'écriture binaire de chaque coordonnée revient à sa forme initiale et le pixel à sa position initiale.

En effet, lorsqu'on applique la fonction photomaton, pour chaque coordonnée (ligne ou colonne) :

- Si elle est paire, on la divise par 2 c'est-à-dire que si le bit de poids faible est 0, alors on le supprime, ce qui revient à faire passer ce 0 de la dernière à la première position dans l'écriture binaire :

$$\underbrace{1.0.1.}_{n-1 \text{ bits}}\boxed{0} \overset{\text{photomaton}}{\Longrightarrow} \boxed{0}\underbrace{1.0.1.}_{n-1 \text{ bits}}$$

- Si elle est impaire, on prend son quotient dans la division euclidienne par 2 et on ajoute 2^{n-1}. Cela revient à faire passer le bit de poids faible égal à 1 de la dernière à la première position dans l'écriture binaire :

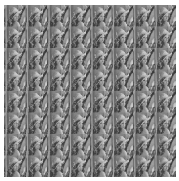
$$\underbrace{\underbrace{1.0.1.}_{n-1 \text{ bits}}}_{\text{quotient par 2}}\boxed{1} \overset{\text{photomaton}}{\Longrightarrow} \underbrace{\boxed{1}\underbrace{0..0}_{n-1 \text{ bits}}}_{2^{n-1}} + \underbrace{1.0.1.}_{n-1 \text{ bits}} = \boxed{1}\underbrace{1.0.1.}_{n-1 \text{ bits}}$$

Source $2^8 \times 2^8$ pixels

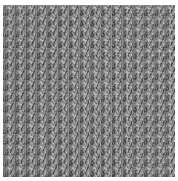
Itération 1

Itération 2

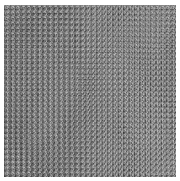
Itération 3



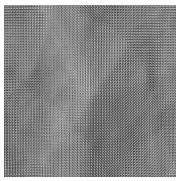
Itération 4



Itération 5



Itération 6



Itération 7

Itération 8