TD : Représentation des images

L'objectif de ce TD est de manipuler des images en Python, dans les formats PBM, PGM et PPM. Ces formats sont des formats *bitmap* (on code la couleur de chaque pixel), écrits en ASCII pour être lisibles et modifiables facilement par un humain. *Le premier exercice est inspiré d'un TP de Pierre Hyvernat.*

Description des formats.
Le format PBM permet de représenter une image en noir et blanc, PGM en niveau de gris, et PPM en couleurs RGB. Le format est le suivant :

- la première ligne contient `P1` (pour PBM), `P2` (pour PGM) ou `P3` (pour PPM) ;
- la seconde contient les dimensions de l'image, sous la forme `largeur hauteur` ;
- pour PGM et PPM, la troisième ligne contient un entier qui est la valeur maximale à utiliser : si c'est par exemple 18, l'image sera codée en 18 niveaux de gris (pour PGM) ou 18 niveaux de couleur (pour PPM) ; cette ligne n'existe pas pour PBM ;
- les lignes suivantes décrivent chaque pixel de l'image, ligne par ligne et colonne par colonne ; pour PPM, chaque pixel nécessite trois entiers ; on peut insérer autant d'espaces qu'on souhaite entre deux pixels, voire ajouter des passages à la ligne intempestifs.
- enfin, le caractère # commence un commentaire (comme en Python).

Remarque : en PBM, le `0` *est le blanc et le* `1` *le noir ; en PGM sur deux niveaux, c'est l'inverse !*

Par exemple, les fichiers ci-dessous représentent les images suivantes :  et  .

```
P1      # Format : PBM
3 4     # Dimensions : largeur hauteur
1 1 1   # Première ligne
1 0 1   # Deuxième ligne
1 0 1   # Troisième ligne
1 1 1   # Quatrième ligne
```

```
P3    # Format : PPM
3 3   # Dimensions
7     # Niveaux de couleurs
0 7 0   7 7 0   7 0 0
4 0 4   0 0 3   2 2 2
0 7 0   7 7 0   7 0 0
```

Manipulation des images en Python.
En Python, les images seront représentées par des matrices (listes de listes) d'entiers ou de triplets d'entiers pour celles en couleur. Par exemple, les fichiers précédents sont représentés en Python par les listes `[[1, 1, 1], [1, 0, 1], [1, 0, 1], [1, 1, 1]]` et `[[(0,7,0), (7,7,0), (7,0,0)], [(4,0,4), (0,0,3), (2,2,2)], [(0,7,0), (7,7,0), (7,0,0)]]`.
On fournit dans `es.py` deux fonctions pour les lectures et écritures de fichier :

- `charger_image(nom)` prend en entrée un nom de fichier et renvoie un triplet `(image, fmt, maxi)` où `image` est la matrice qui représente l'image, `fmt` vaut `'PBM'`, `'PGM'` ou `'PPM'`, `maxi` est la valeur maximale (0 pour les PBM) ;
- `sauver_image(image, fmt, nom, maxi = 0)` prend en entrée une image (matrice), un format (`'PBM'`, `'PGM'` ou `'PPM'`) et un nom de fichier (sans extension) et écrit le fichier `nom.ext` (avec la bonne extension) représentant l'`image`. Le dernier paramètre optionel permet de fournir la valeur maximale à utiliser : s'il n'est pas présent, on utilise la valeur maximale trouvée dans la matrice.

Par exemple, on peut charger le fichier précédent, puis ré-enregistrer l'image en niveaux de gris de la manière suivante (remarquer que les couleurs sont inversées, à cause de la remarque précédente) :

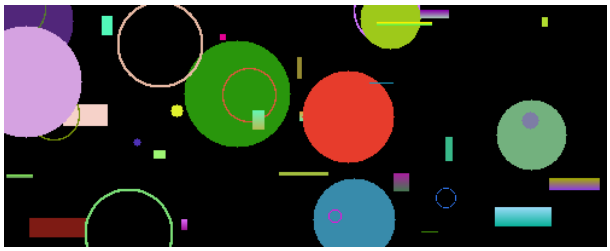
```
>>> from es import *
>>> image, fmt, maxi = charger_image('exemple.pbm')
>>> image
[[0,0,0],[0,1,0],[0,0,0]]
>>> sauver_image(image, 'PGM', 'engris', 2)
```

Enfin, pour visualiser les images, on pourra utiliser le logiciel `display` (Clic-droit > Ouvrir avec une autre application > Utiliser une commande personnalisée > Entrer `display`).

Exercice 1. *Dessins*

1. Créer trois fichiers pour obtenir les images ci-dessous.

2. i. Écrire les fonctions `hauteur_image(image)` et `largeur_image(image)` qui prennent en entrée une matrice et renvoient respectivement la hauteur et la largeur, en pixel, de l'image correspondante.
   ii. Écrire une fonction `initialise(largeur, hauteur, couleur)` qui crée la matrice de bonnes dimensions qui ne consiste qu'en des pixels de la couleur fournie. *Tester la fonction en enregistrant une image produite dans un fichier.*
   iii. Écrire une fonction `couleur_pixel(image, i, j)` qui renvoie la couleur du pixel de coordonnées (i, j).
   iv. Écrire une fonction `change_pixel(image, i, j, couleur)` qui met le pixel de coordonnées (i, j) à la couleur fournie. *Cette fonction ne renvoie rien.*
3. i. Écrire une fonction `rectangle(image, x0, y0, largeur, hauteur, couleur)` qui ajoute dans l'`image` un rectangle de dimensions `largeur` $\times$ `hauteur` de la `couleur` fournie, dont le pixel en haut à gauche est au coordonnées `(x0,y0)`. *Remarques : cette fonction (et les suivantes) doit modifier une image existante, ne pas en créer une. D'autre part, il faut qu'elle fonctionne même si le rectangle demandé dépasse de l'image : dans ce cas, il est simplement tronqué.*
   ii. Écrire une fonction `rectangle_degrade(image, x0, y0, largeur, hauteur, couleur_bas, couleur_haut)` qui effectue la même tâche, mais où le rectangle est en dégradé de couleurs, entre `couleur_bas` et `couleur_haut`. *On pourra supposer que le format d'image est PPM.*
4. i. Écrire une fonction `disque(image, x0, y0, rayon, couleur)` qui ajoute dans l'`image` un disque de centre `(x0,y0)` de `rayon` et `couleur` fournis.
   ii. Écrire une fonction `cercle(image, x0, y0, rayon, couleur)` qui ajoute dans l'`image` un cercle.
5. À l'aide de la fonction `randint` du module `random`[1], écrire une fonction qui permet de créer un *tableau aléatoire* comme celui ci-dessous.



Exercice 2. *Compression d'images*

Le but de l'exercice est de voir quelques idées très basiques de compression d'images, avec ou sans perte. On utilisera comme exemples des tableaux aléatoires de l'exercice précédent.

1. i. Écrire une fonction `niveaux_de_gris(image, maxi)` qui transforme une image couleur (PPM) en niveaux de gris (PGM), `maxi` étant la valeur maximale de l'image de départ et d'arrivée.
   ii. Appliquer la fonction sur divers tableaux et comparer les tailles de fichiers.
2. i. Écrire une fonction `reduction_couleurs(image, maxi, maxi_nv)` qui prend en entrée une `image` couleur représentée avec `maxi` couleurs, et crée une image ressemblante mais codée sur `maxi_nv` couleurs.
   ii. Appliquer la fonction sur divers tableaux et comparer les tailles de fichiers. Que faudrait-il faire pour que la taille diminue vraiment ?

On définit trois nouveaux formats de fichiers (PBMX, PGMX et PPMX[2]) qui sont des versions compressées des formats précédents. Au lieu de stocker les pixels les uns à la suite des autres, on stocke des suites de pixels

1. L'appel à `randint(a,b)` renvoie un entier aléatoire uniforme dans l'intervalle $[a, b]$.
2. Contrairement aux formats PPM, PGM et PPM qui sont standard, les formats compressés sont une invention pour ce TP.

contigus de même couleur avec un entier (le nombre de pixels) suivi de la couleur. Par exemple, pour un fichier PBM, la ligne

```
0 0 0 0 0 0 1 1 1 1 0 0 0
```

devient en format PBMX

```
7 0 4 1 3 0
```

Le fichier `es.py` permet également de sauvegarder et charger des images compressée.
En Python, une image compressée sera représentée par une matrice de couples (longueur, couleur) : la ligne précédente est par exemple `[(7,0), (4,1), (3,0)]`.

3. **i.** Écrire une fonction `decompression(image_compressee)` qui prend en entrée une image en représentation compressée, et renvoie sa représentation standard.
ii. Écrire une fonction `compression(image)` qui prend en entrée une image en représentation standard et renvoie sa représentation compressée.
iii. Tester vos fonctions en vérifiant que `compression(decompression(image)) == image`.
iv. Appliquer la fonction sur divers tableaux et comparer les tailles de fichier.
v. À l'aide du module `random`, créer une image faite de pixels aléatoires et tester la compression sur cette image. Qu'observe-t-on sur les tailles de fichier ?

Exercice 3. *Images vectorielles*
On définit trois nouveaux format de fichiers (PBMV, PGMV, PPMV[3]) pour représenter des images vectorielles basiques. Par exemple, le fichier suivant représente une image de dimension 30×40, de fond noir, avec un rectangle rouge, un disque vert et un cercle bleu.

```
V3                          # Format PPMV
30 40                       # Dimensions
255                         # Couleur maximale
0 0 0                       # Couleur de fond
rectangle 12 17 6 8 255 0 0 # rectangle d'origine (12,17), de dim. 6×8,
                            # de couleur (255,0,0)
disque 3 5 7 0 255 0        # disque de centre (3,5), de rayon 7, de couleur (0,255,0)
cercle 22 2 5 0 0 255       # cercle de centre (22,2), de rayon 5, de couleur (0,0,255)
```

En Python, la figure correspondante est représentée par le quadruplet `(30, 40, (0,0,0), objets)` où `objets` est la liste `[('rectangle',12,17,6,8,(255,0,0)), ('disque',3,5,7,(0,255,0)), ('cercle',22,2,5,(0,0,255)]`.
Les fonctions fournies dans `es.py` permettent également de sauver et charger des images vectorielles.

1. Écrire une fonction qui crée une image vectorielle aléatoire, de la même façon que dans la dernière question de l'exercice **1**.
2. **i.** Écrire une fonction `rasterize(image_vect, zoom)` qui prend en entrée une image vectorielle et un niveau de `zoom` (entier) et renvoie une image standard (format PPM si c'est en couleur) correspondant à l'image vectorielle, en appliquant le `zoom` : si l'image vectorielle a dimensions `L` et `H`, l'image standard aura dimensions `L*zoom` et `H*zoom`.
ii. Créer un tableau aléatoire sous forme vectorielle, et ses versions non vectorielles à plusieurs niveau de zoom. Comparer les rendus, et les tailles de fichiers.

3. Même remarque !