Séance 8 : Ensembles, dictionnaires et matrices

L1 – Université Côte d'Azur

Dans ce TP, vous allez manipuler des images aux formats `PBM`, `PGM` et `PPM`. Ces formats permettent très facilement de lire et modifier une (petite) image avec un éditeur de texte. Pour commencer, récupérez l'archive `tp8.zip` à l'adresse : https://upinfo.univ-cotedazur.fr/~obaldellon/L1/py/tp8/tp8.zip, décompressez-la dans le répertoire qui contient tous vos TP Python, puis allez dans le répertoire tp8. Vous devez y trouver les fichiers

```
exemple.pbm
space-invader-pieuvre.pbm
space-invader-soucoupe.pbm
```

```
pnm.py
chat.py
```

Le format PBM

Un fichier PBM (portable bitmap) est un fichier qui contient une image noir et blanc pixelisée.

```
# fichier exemple.pbm
P1
# indique le format pbm
5   # indique la largueur de l'image
3   # indique la hauteur de l'image
1 1 1 1 1  # une premiere ligne noire
0 0 0 0 0  # une deuxieme ligne blanche
1 1 1      # debut troisieme ligne
1 1        # fin troisieme ligne
```

En plus d'une extension `.pbm`, un fichier PBM doit commencer par `P1`. Ensuite, le fichier contient seulement des nombres et des commentaires. Comme en Python, tout ce qui suit un `#` sur une ligne est un commentaire. Les nombres sont séparés par des espaces ou des passages à la ligne. Les deux premiers nombres sont la largeur et la hauteur de l'image. Les nombres suivants codent les couleurs de chaque pixel, ligne par ligne et de gauche à droite, et prennent la valeur 0 pour un pixel blanc et 1 pour un pixel noir.

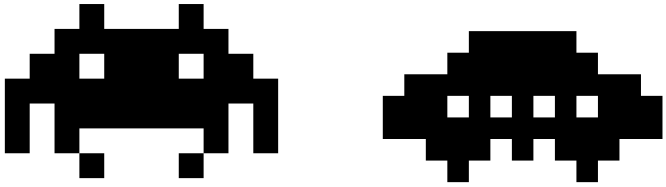
Exercice 1 — Lire et modifier un fichier PBM (⋆)

1. Créez un fichier `tp8.py` dans le répertoire contenant tous les fichiers extraits de l'archive. Affichez les deux images (`space-invader-pieuvre.pbm` et `space-invader-soucoupe.pbm`) en utilisant la fonction `voir_fichier` du module `pnm.py`. Vous pouvez vous inspirer de l'exemple ci-dessous.

```python
from pnm import * # À vous de modifier la suite
voir_fichier('space-invader-pieuvre.pbm' , quadrillage=True)
voir_fichier('space-invader-pieuvre.pbm')
```

```python
from pnm import *
voir_fichier('space-invader-pieuvre.pbm',quadrillage=False)
voir_fichier('space-invader-soucoupe.pbm',quadrillage=False)
```

   Vous devriez voir, successivement, les images suivantes :



2. À l'aide de Thonny (Menu Fichier, Ouvrir, et « tous les fichiers» ou « all files » en bas à droite), ouvrez le fichier : `space-invader-pieuvre.pbm`. Quelles sont les dimensions de l'image ? Si vous remplacez le 9 par un 8, comment est modifiée l'image ? Et le 13 par un 12 ? Comment faut-il modifier le fichier pour obtenir l'image ci-dessous ?

   (a) Si on remplace 9 par 8 la dernière ligne n'est pas prise en compte

   (b) Si on remplace 13 par 12 chaque ligne se décale d'un pixel et le dernier pixel de chaque ligne (le 13) devient le premier pixel de la ligne suivante.

   (c) Il suffit de remplacer le 0 correspondant au premier œil par un 1.

3. Ajoutez au fichier `tp8.py` le code ci-dessous et exécutez-le. À quoi correspond le contenu de la matrice M ?

```
M = fichier_vers_matrice('space-invader-pieuvre.pbm')
print(M)
```

   `M[i][j]` contient `True` si le pixel `(i,j)` est noir ; `False` si le pixel `(i,j)` est blanc.

4. Complétez les `...` dans le code ci-dessous afin de crée un fichier `space-invader-pieuvre2.pbm` qui contiendra une pieuvre qui ferme son œil gauche comme vu plus haut.

```
M = fichier_vers_matrice('space-invader-pieuvre.pbm')
M[ ... ][ ... ] = ....
matrice_vers_fichier(M,'space-invader-pieuvre2.pbm')
voir_fichier('space-invader-pieuvre2.pbm')
```

   Attention, il y a un bord blanc dans l'image `space-invader-pieuvre.pbm`.

```
M = fichier_vers_matrice('space-invader-pieuvre.pbm')
M[3][4] = True
matrice_vers_fichier(M, 'space-invader-pieuvre2.pbm')
voir_fichier('space-invader-pieuvre2.pbm')
```

Exercice 2 — Afficher une image PBM en mode texte (⋆ −⋆⋆)

On rappelle que, dans ce TP, vous pouvez définir et utiliser les deux fonctions vues en cours :

- `dimensions(M)` qui renvoie un couple `(n,m)` correspondant au nombre de lignes et de colonnes.
- `matrice_vide(n,m)` qui renvoie une matrice de dimensions $n \times m$ et ne contenant que des `None`.

Dans le fichier `tp8.py`, écrivez une fonction `affiche_matrice_booléens(M,plein,vide)` qui prend en argument une matrice `M`, un caractère `plein` et un caractère `vide` et qui affiche la matrice sous forme rectangulaire en remplaçant `True` par `plein` et `False` par `vide`. Par exemple, si `M` est la matrice obtenue en lisant `space-invader-pieuvre.pbm`, `affiche_matrice(M,'#','.')` affichera le texte ci-dessous.

```
............
....#...#....
...#######...
..##.###.##..
.###########.
.#.#######.#.
.#.#.....#.#.
....#...#....
............
```

On réutilise la fonction dimensions vu en cours.

```python
def dimensions(M):
    return (len(M),len(M[0]))

def affiche_matrice_booleens(M, plein, vide):
    (n,m)=dimensions(M)
    for i in range(n):
        for j in range(m):
            if M[i][j]:
                print(plein, sep='', end='')
            else:
                print(vide, sep='', end='')
        print()
```

On peut éviter le `if` grâce à un dictionnaire

```python
def affiche_matrice_booleens2(M, plein, vide):
    dico = { True:plein, False:vide }
    (n,m)=dimensions(M)
    for i in range(n):
        for j in range(m):
            print(dico[ M[i][j] ],end='')
        print()
```

Exercice 3 — Une armée tentaculaire (⋆⋆)

On veut maintenant construire l'image suivante

On va pour cela utiliser un accumulateur pour construire chaque ligne puis faire leur somme par concaténation. Cependant, l'opération de concaténation de matrices n'est pas prédéfinie en Python, et il va falloir commencer par la définir.

1. Écrivez une fonction `ajoute_horizontal(M1,M2)` qui prend en arguments deux matrices M_1, M_2 de dimensions respectives l_1 colonnes × h lignes et l_2 colonnes × h lignes et qui renvoie la matrice dont les dimensions sont $l_1 + l_2$ colonnes × h lignes obtenue en concaténant M_2 à droite de M_1. Testez votre fonction en exécutant le code ci-dessous puis en ouvrant le fichier `space-invader-3-pieuvres.pbm`.

```python
from pnm import *

def ajoute_horizontal(M1,M2) :
    # votre code ici

M = fichier_vers_matrice('space-invader-pieuvre.pbm')
M2 = ajoute_horizontal(M , M)
M3 = ajoute_horizontal(M2 , M)
matrice_vers_fichier(M3 , 'space-invader-3-pieuvres.pbm')
```

On crée une matrice remplie de False et puis on remplit les bonnes valeurs. C'est la façon classique et recommandée pour travailler avec des matrices.

```python
def matrice_false(n,m):
    # Il existe une autre version plus longue dans le cours : matrice_nulle
    # Au lieu de remplir la matrice de 0, nous allons la remplir de False
    return [ [False] * (m) for i in range(n)]

def ajoute_horizontal(M1 , M2):
    """Construit une nouvelle matrice constituée des colonnes de M1 et à
        droite de celles de M2 (erreur si M1 et M2 n'ont pas le même
        nombre de lignes).
    """
    (n1, m1) = dimensions(M1)
    (n2, m2) = dimensions(M2)
    assert n1 == n2
    n=n1
    M = matrice_false(n,m1+m2)
    for i in range(n):
        for j in range(m1):
            M[i][j] = M1[i][j]
        for j in range(m2) :
            M[i][j+m1] = M2[i][j]
    return M
```

On aurait aussi pu remarquer que l'on peut obtenir chaque ligne M[i] de M en concaténant M1[i] + M2[i], ce qui permet le code en une ligne suivant :

```python
def ajoute_horizontal(M1 , M2):
    assert(len(M1) == len(M2))
    return  [ M1[i] + M2[i] for i in range(len(M1)) ]
```

2. Écrivez une fonction `répète_horizontal(M,n)` qui prend en argument une matrice M et un entier $n > 0$ et qui renvoie la matrice M répétée n fois horizontalement. En utilisant cette fonction, créez un fichier que vous nommerez `space-invader-20-pieuvres.pbm` contenant une rangée de 20 pieuvres.

```python
def répète_horizontal(M, n):
    matrice = M
    for i in range(1,n):
        matrice = ajoute_horizontal(matrice,M)
    return matrice
```

3. Écrivez une fonction `ajoute_vertical(M1,M2)` qui prend en arguments une matrice M_1 de dimension $l_1 \times h_1$ et une matrice M_2 de dimension $l_2 \times h_2$ et qui renvoie une matrice M de dimension $\max(l_1, l_2) \times (h_1 + h_2)$ obtenue en concaténant M_2 en dessous de M_1 en les alignant sur la gauche.
   Optionnel : au lieu d'aligner à gauche, on pourra centrer les matrices l'une sous l'autre.
   Testez votre fonction avec le code suivant.

```
M1 = fichier_vers_matrice('space-invader-pieuvre.pbm')
M2 = fichier_vers_matrice('space-invader-soucoupe.pbm')
M3 = ajoute_vertical(M1 , M2)
matrice_vers_fichier(M3,'space-invader-vertical.pbm')
```

La version à gauche

```
def ajoute_vertical(M1,M2):
    (n1,m1) = dimensions(M1)
    (n2,m2) = dimensions(M2)
    n = n1+n2
    m = max(m1,m2)
    # On crée une matrice de la bonne taille
    M=matrice_false(n,m)
    # On complète le haut de la matrice M avec M1
    for i in range(n1):
        for j in range(m1): # m1<m donc M[i][j] bien définie
            M[i][j] = M1[i][j]
    # On complète le bas de la matrice M avec M1
    for i in range(n2): # i+n1 <= n2+n1 <= n donc M[i+n1] bien définie
        for j in range(m2): # m2<m donc M[i][j] bien définie
            M[i+n1][j] = M2[i][j]
    return M
```

La version centrée

```
def ajoute_vertical_centré(M1,M2):
    (n1,m1) = dimensions(M1)
    (n2,m2) = dimensions(M2)
    n = n1+n2
    m = max(m1,m2)
    M=matrice_false(n,m)
    delta1 = (m-m1)//2 # L'écart à ajouter au début pour centrer la matrice Mk
    delta2 = (m-m2)//2 # Remarque si Mk est la plus grande, deltak sera nul (k=1 ou 2)
    for i in range(n1):
        for j in range(m1):
            M[i][j+delta1] = M1[i][j]
    for i in range(n2):
        for j in range(m2):
            M[i+n1][j+delta2] = M2[i][j]
    return M

M1 = fichier_vers_matrice('space-invader-pieuvre.pbm')
M2 = fichier_vers_matrice('space-invader-soucoupe.pbm')
M3 = ajoute_vertical_centré(M1 , M2)
matrice_vers_fichier(M3,'space-invader-vertical-centre.pbm')
voir_fichier('space-invader-vertical-centre.pbm')
```

4. Créez le fichier `space-invader-army.pbm` qui contient l'image représentée plus haut.

```python
def armée(nb_lignes, nb_pieuvres, nb_soucoupes):
    M1 = fichier_vers_matrice('space-invader-pieuvre.pbm')
    M2 = fichier_vers_matrice('space-invader-soucoupe.pbm')
    M1 = répète_horizontal(M1, nb_pieuvres)
    M2 = répète_horizontal(M2, nb_soucoupes)
    M = M1
    for i in range(1, nb_lignes): # On commence à 1, car au début  M=M1
        if i%2 == 0:
            M = ajoute_vertical_centré(M, M1)
        else:
            M = ajoute_vertical_centré(M, M2)
    return M
```

Exercice 4 — Un peu de couleurs dans ce monde de brutes (⋆)

Pour l'instant, nous avons travaillé uniquement avec le format PBM ou les pixels sont noirs ou blancs. Il existe deux autres formats de la même famille, le format PGM en nuance de gris et le format PPM en couleur. C'est ce dernier que nous allons utiliser dans cette exercice. L'ensemble de ces trois formats est parfois désigné sous le nom de PNM.

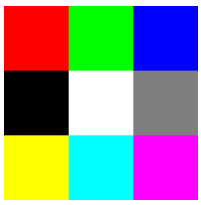
Une matrice correspondant au format PPM est une matrice contenant des triplets `(r,g,b)` où `r`, `g` et `b` sont des entiers entre 0 et 255 compris.

Dans un nouveau fichier, créez une matrice `M` de taile 3×3, dont la première ligne contient du rouge, du vert et du bleu, la seconde du noir du blanc et du gris et la dernière du jaune, du cyan et du magenta. Nous avons vu dans le cours 4 comment coder des couleurs via des triplets.

```python
from pnm import *

M=[...] # À compléter
voir_matrice(M)
```

La fonction `voir_matrice` permet d'afficher l'image correspondant à la matrice. Vous devriez donc obtenir l'image ci-dessous.



```python
rouge   = (255,  0,  0)
vert    = (  0,255,  0)
bleu    = (  0,  0,255)
noir    = (  0,  0,  0)
blanc   = (255,255,255)
gris    = (127,127,127)
jaune   = (255,255,  0)
cyan    = (  0,255,255)
magenta = (255,  0,255)
M = [ [rouge,vert,bleu], [noir,blanc,gris], [jaune, cyan,magenta] ]
voir_matrice(M)
```

Exercice 5 — Une photo de chat (⋆)

Nous arrivons à la fin de l'année et pour vous remercier de votre assiduité, nous vous offrons une photo de chat. Cette photo est définie dans une matrice `chat_mignon` dans le fichier `chat.py` (ne cherchez pas à ouvrir ce fichier, il est très volumineux).

```
from pnm import *
from chat import chat_mignon

voir_matrice(chat_mignon)
```

Malheureusement, la matrice est corrompue, chaque valeur ayant été multipliée par 2. Par exemple une couleur `(100,255,3)` est codée dans la matrice `chat_mignon` par `(200,510,6)`.

Créez une nouvelle matrice en corrigeant les valeurs des couleurs et affichez l'image correspondante.

```
from pnm import *
from chat import chat_mignon

(n,m)=dimensions(chat_mignon)
python_mignon = matrice_false(n,m)

for i in range(n):
    for j in range(m):
        (r,v,b) = chat_mignon[i][j]
        python_mignon[i][j] = (r//2, v//2, b//2)

voir_matrice(python_mignon)
```