

## Séance 7 : MODULES ET TYPES ABSTRAITS

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 `tp7.zip` à l'adresse : <https://upinfo.univ-cotedazur.fr/~obaldellon/L1/bil/tp7/tp7.zip>, décompressez-la dans le répertoire qui contient tous vos TP Python, puis allez dans le répertoire `tp7`. Vous devez y trouver les fichiers

<code>exemple.pbm</code>	<code>pnm.py</code>
<code>space-invader-pieuvre.pbm</code>	<code>matrice.py</code>
<code>space-invader-soucoupe.pbm</code>	<code>chat.py</code>

### 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 largeur de l'image
3  # indique la hauteur de l'image
1 1 1 1 1  # une première ligne noire
0 0 0 0 0  # une deuxième ligne blanche
1 1 1      # début troisième ligne
1 1        # fin troisième 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.

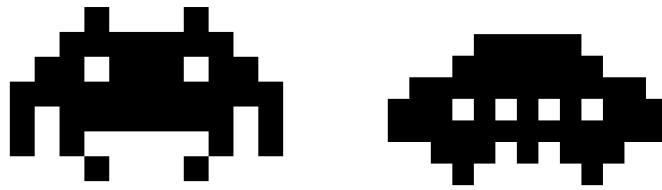
### Exercice 1 – Lire et modifier un fichier PBM

1. Créez un fichier `tp7.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.

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

```
1 from pnm import *
2 voir_fichier('space-invader-pieuvre.pbm', quadrillage=False)
3 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 `tp7.py` le code ci-dessous et exécutez-le. À quoi correspond le contenu de la matrice `M` ?

```
1 M = fichier_vers_matrice('space-invader-pieuvre.pbm')
2 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 créer un fichier `space-invader-pieuvre2.pbm` qui contiendra une pieuvre qui ferme son œil gauche comme vu plus haut.

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

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

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

## Exercice 2 — Afficher une image PBM en mode texte

Dans ce TP, vous pouvez et devez utiliser les deux fonctions vues en cours et présentes dans le fichier `matrice.py` :

- `dimensions(M)` qui renvoie un couple  $(n, m)$  correspondant au nombre de lignes et de colonnes.
- `nouvelle_matrice(n, m)` qui renvoie une matrice de dimensions  $n \times m$  et ne contenant que des `False`.

Dans le fichier `tp7.py`, écrivez une fonction `affiche_matrice_booléens(M, plein, vide)` qui prend en arguments 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 et qui se trouve dans le fichier *matrice.py*

```

1 def affiche_matrice_booleens(M, plein, vide):
2     (n,m)=dimensions(M)
3     for i in range(n):
4         for j in range(m):
5             if M[i][j]:
6                 print(plein, sep='', end='')
7             else:
8                 print(vide, sep='', end='')
9     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.

Dans cet exercice, on créera à chaque fois une matrice de la bonne taille que l'on remplira dans un second temps avec les bonnes valeurs. Il est interdit d'utiliser la méthode `append` ou la concaténation sur les listes.

1. Écrivez une fonction `ajoute_horizontale(M1,M2)` qui prend en arguments deux matrices  $M_1, M_2$  de dimensions respectives  $n$  lignes  $\times m_1$  colonnes et  $n$  lignes  $\times m_2$  colonnes et qui renvoie la matrice dont les dimensions sont  $n$  lignes  $\times m_1 + m_2$  colonnes obtenue en concaténant  $M_2$  à droite de  $M_1$ . On créera une matrice de la bonne taille que l'on remplira dans un second temps avec les bonnes valeurs. Testez votre fonction en exécutant le code ci-dessous puis en ouvrant le fichier `space-invader-3-pieuvres.pbm`.

```

1 from pnm import *
2
3 def ajoute_horizontale(M1,M2) :
4     # votre code ici
5
6 M = fichier_vers_matrice('space-invader-pieuvre.pbm')
7 M2 = ajoute_horizontale(M , M)
8 M3 = ajoute_horizontale(M2 , M)
9 matrice_vers_fichier(M3 , 'space-invader-3-pieuvres.pbm')

```

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

```

1 def ajoute_horizontal(M1 , M2):
2     """Construit une nouvelle matrice constituée des colonnes de M1 et à
3         droite de celles de M2 (erreur si M1 et M2 n'ont pas le même
4         nombre de lignes).
5     """
6     (n1, m1) = dimensions(M1)
7     (n2, m2) = dimensions(M2)
8     assert n1 == n2
9     n=n1
10    M = nouvelle_matrice(n,m1+m2)
11    for i in range(n):
12        for j in range(m1):
13            M[i][j] = M1[i][j]
14        for j in range(m2) :
15            M[i][j+m1] = M2[i][j]
16    return M

```

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

```

1 def répète_horizontal(M, k):
2     matrice = M
3     for i in range(1,k):
4         matrice = ajoute_horizontal(matrice,M)
5     return matrice

```

3. Écrivez une fonction `ajoute_vertical(M1,M2)` qui prend en arguments une matrice  $M_1$  de dimension  $n_1 \times m_1$  et une matrice  $M_2$  de dimension  $n_2 \times m_2$  et qui renvoie une matrice  $M$  de dimension  $(n_1 + n_2) \times \max(m_1, m_2)$  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.

```

1 M1 = fichier_vers_matrice('space-invader-pieuvre.pbm')
2 M2 = fichier_vers_matrice('space-invader-soucoupe.pbm')
3 M3 = ajoute_vertical(M1 , M2)
4 matrice_vers_fichier(M3, 'space-invader-vertical.pbm')

```

*La version à gauche*

```

1 def ajoute_vertical(M1,M2):
2     (n1,m1) = dimensions(M1)
3     (n2,m2) = dimensions(M2)
4     n = n1+n2
5     m = max(m1,m2)
6     # On crée une matrice de la bonne taille
7     M=nouvelle_matrice(n,m)
8     # On complète le haut de la matrice M avec M1
9     for i in range(n1):
10        for j in range(m1): # m1<m donc M[i][j] bien définie
11            M[i][j] = M1[i][j]
12    # On complète le bas de la matrice M avec M2
13    for i in range(n2): # i+n1 <= n2+n1 <= n donc M[i+n1] bien définie
14        for j in range(m2): # m2<m donc M[i][j] bien définie
15            M[i+n1][j] = M2[i][j]
16    return M

```

*La version centrée*

```

1 def ajoute_vertical_centré(M1,M2):
2     (n1,m1) = dimensions(M1)
3     (n2,m2) = dimensions(M2)
4     n = n1+n2
5     m = max(m1,m2)
6     M=nouvelle_matrice(n,m)
7     delta1 = (m-m1)//2 # L'écart à ajouter au début pour centrer la matrice Mk
8     delta2 = (m-m2)//2 # Remarque si Mk est la plus grande, deltak sera nul (k=1 ou 2)
9     for i in range(n1):
10         for j in range(m1):
11             M[i][j+delta1] = M1[i][j]
12     for i in range(n2):
13         for j in range(m2):
14             M[i+n1][j+delta2] = M2[i][j]
15     return M
16
17 M1 = fichier_vers_matrice('space-invader-pieuvre.pbm')
18 M2 = fichier_vers_matrice('space-invader-soucoupe.pbm')
19 M3 = ajoute_vertical_centré(M1 , M2)
20 matrice_vers_fichier(M3, 'space-invader-vertical-centre.pbm')
21 voir_fichier('space-invader-vertical-centre.pbm')

```

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

```

1 def armée(nb_lignes, nb_pieuvres, nb_soucoupes):
2     M1 = fichier_vers_matrice('space-invader-pieuvre.pbm')
3     M2 = fichier_vers_matrice('space-invader-soucoupe.pbm')
4     M1 = répète_horizontale(M1, nb_pieuvres)
5     M2 = répète_horizontale(M2, nb_soucoupes)
6     M = M1
7     for i in range(1, nb_lignes): # On commence à 1, car au début M=M1
8         if i%2 == 0:
9             M = ajoute_vertical_centré(M, M1)
10        else:
11            M = ajoute_vertical_centré(M, M2)
12    return M

```

#### Exercice 4 — Un peu de couleurs dans ce monde de brutes

Pour l'instant, nous avons travaillé uniquement avec le format PBM où 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 cet exercice. L'ensemble de ces trois formats est parfois désigné sous le nom de PNM.

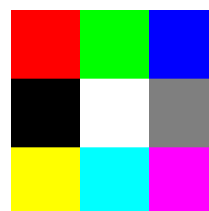
Une matrice correspondant au format PPM est une matrice contenant des triplets  $(r, g, b)$  où  $r, g$  et  $b$  sont des entiers entre 0 et 255 compris.

Dans un nouveau fichier, créez une matrice  $M$  de taille  $3 \times 3$ , 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.

```

1 from pnm import *
2
3 M=[...] # À compléter
4 voir_matrice(M)

```



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

```

1 rouge = (255, 0, 0)
2 vert = ( 0, 255, 0)
3 bleu = ( 0, 0, 255)
4 noir = ( 0, 0, 0)
5 blanc = (255, 255, 255)
6 gris = (127, 127, 127)
7 jaune = (255, 255, 0)
8 cyan = ( 0, 255, 255)
9 magenta = (255, 0, 255)
10 M = [ [rouge, vert, bleu], [noir, blanc, gris], [jaune, cyan, magenta] ]
11 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).

```

1 from pnm import *
2 from chat import chat_mignon
3
4 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).

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

```

1 from pnm import *
2 from chat import chat_mignon
3
4 (n,m)=dimensions(chat_mignon)
5 python_mignon = nouvelle_matrice(n,m)
6
7 for i in range(n):
8     for j in range(m):
9         (r,v,b) = chat_mignon[i][j]
10        python_mignon[i][j] = (r//2, v//2, b//2)
11
12 voir_matrice(python_mignon)

```

2. Modifier l'image pour qu'elle devienne en noir et blanc. (indice : pour obtenir un pixel gris, il faut que le rouge, le vert et le bleu soit en proportions égales.)

```

1 (n,m)=dimensions(python_mignon)
2 for i in range(n):
3     for j in range(m):
4         (r,v,b) = python_mignon[i][j]
5         moyenne = (r+v+b)//3
6         python_mignon[i][j] = (moyenne, moyenne, moyenne)
7
8 voir_matrice(python_mignon)

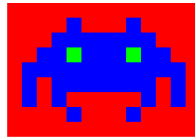
```

### Exercice 6 — Le pull moche de Noël

Pour se préparer à l'esprit de Noël, on souhaite créer un motif pour un pull à tricoter pour les fêtes de fin d'année.

1. Écrire une fonction `coloriser_matrice(M, corps, yeux, fond)` qui à partir d'une matrice en noir et blanc renvoie une nouvelle matrice colorisée. On suppose que `corps`, `yeux`, `fond` sont trois couleurs au format RGB

(triplet). Le noir sera alors remplacé par la couleur corps, le blanc par la couleur fond et les cases blanches entourées par 8 cases noires seront en couleur yeux.

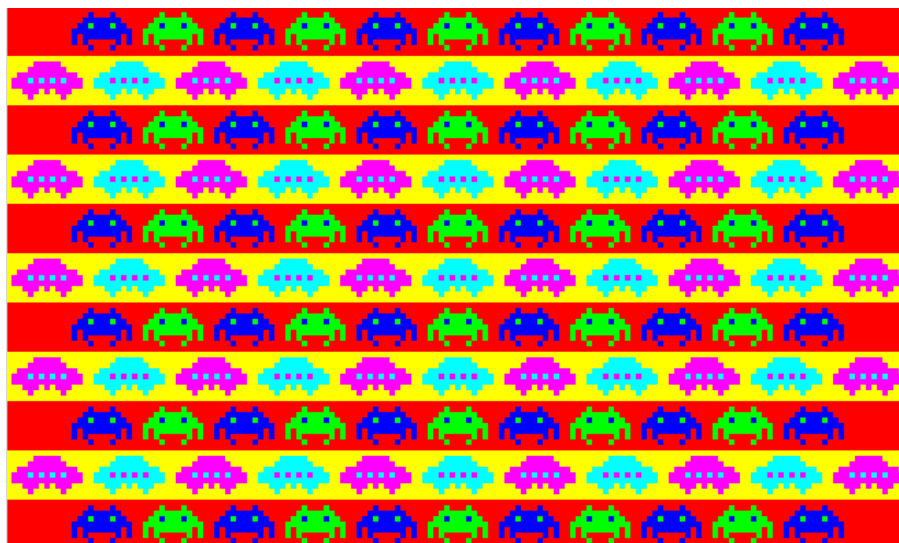


```

1 def est_yeux(i,j,matrice):
2     (n,m) = dimensions(matrice)
3     if matrice[i][j]: # si le pixel est noir
4         return False
5     if i in [0, n-1] or j in [0, m-1]: # si le pixel est sur un bord
6         return False
7
8     for a in range(i-1, i+2):
9         for b in range(j-1, j+2):
10            if (a,b) != (i,j): # on ignore le pixel i,j
11                if not matrice[a][b]: # si le pixel est blanc
12                    return False
13
14     return True
15
16 def coloriser_matrice(fichier,corps, yeux, fond):
17     M = fichier_vers_matrice(fichier)
18     (n,m) = dimensions(M)
19     C = nouvelle_matrice(n,m)
20     for i in range(n):
21         for j in range(m):
22             if est_yeux(i,j,M):
23                 C[i][j] = yeux
24             elif M[i][j]:
25                 C[i][j] = corps
26             else:
27                 C[i][j] = fond
28     return C

```

2. En déduire un programme qui affiche l'image ci-dessous :



On commence par modifier la fonction `ajoute_vertical_centré` pour colorier les bords de la plus petite image (sinon, les bords seraient blancs).

```

1 def remplir_matrice(M,x):
2     (n,m)=dimensions(M)
3     for i in range(n):
4         for j in range(m):
5             M[i][j] = x
6
7 # Je rajoute deux paramètres x et y pour indiquer les couleurs de fond
8 # de M1 (de couleur x) et M2 (de couleur y).
9 def ajoute_vertical_centré(M1,M2,x,y):
10     (n1,m1) = dimensions(M1)
11     (n2,m2) = dimensions(M2)
12     n = n1+n2
13     m = max(m1,m2)
14     M=nouvelle_matrice(n,m)
15     if m==m1:
16         remplir_matrice(M, y)
17     else:
18         remplir_matrice(M, x)
19     delta1 = (m-m1)//2
20     delta2 = (m-m2)//2
21     for i in range(n1):
22         for j in range(m1):
23             M[i][j+delta1] = M1[i][j]
24     for i in range(n2):
25         for j in range(m2):
26             M[i+n1][j+delta2] = M2[i][j]
27     return M

```

On peut alors générer notre motif de pull moche :

```

1 P1 = coloriser_matrice('space-invader-pieuvre.pbm', bleu, vert, rouge)
2 P2 = coloriser_matrice('space-invader-pieuvre.pbm', vert, bleu, rouge)
3 S1 = coloriser_matrice('space-invader-soucoupe.pbm', magenta, cyan, jaune)
4 S2 = coloriser_matrice('space-invader-soucoupe.pbm', cyan, magenta, jaune)
5
6 P = P1 # la ligne des pieuvres
7 S = S1 # la ligne des soucoupes
8 for i in range(10):
9     if i%2==0:
10         P = ajoute_horizontal(P,P2)
11         S = ajoute_horizontal(S,S2)
12     else:
13         P = ajoute_horizontal(P,P1)
14         S = ajoute_horizontal(S,S1)
15
16 M = P
17 for i in range(10):
18     if i%2==0:
19         M = ajoute_vertical_centré(M,S,rouge, jaune)
20     else:
21         M = ajoute_vertical_centré(M,P,jaune,rouge)
22
23 voir_matrice(M)
24

```