1 Listes

Exercice 1.1 (Affichage) On donne la liste suivante : `L=[200,123,14,18,6,19]`

1. Que renvoie les intructions suivantes ?
   (a) `len(L)` (b) `L[0]` (c) `L[3]` (d) `L[6]`
2. Que devient la liste `L` si on saisit l'instruction `L[1]=12` ?

Exercice 1.2 (Seuil) Ecrire une fonction `SeuilListe(L,val)` prenant en argument une liste numérique `L` et une valeur `val` et renvoyant une liste dans laquelle tous les éléments de `L` supérieurs ou égaux à `val` ont été remplacés par 0.
Par exemple, `SeuilListe([1,12,15,3],10)` renvoie la liste `[1,0,0,3]`.

Exercice 1.3 (Plus petit et plus grand élément) Sans utiliser les fonctions `min` et `max`, écrire deux fonctions `PlusPetitElement(L)` et `PlusGrandElement(L)` prenant pour argument une liste numérique `L` et renvoyant respectivement la valeur du plus petit et du plus grand élément de cette liste.

Exercice 1.4 (Echange) Ecrire une fonction `EchangeElement(L,i,j)` prenant en arguments une liste `L` et deux entiers naturels `i` et `j` et renvoyant une liste où les éléments d'indices `i` et `j` de `L`, s'ils existent, ont été permutés.

Exercice 1.5 (Listes palindromes) Une liste numérique est dite *liste palindrome* si elle se lit dans les deux sens. Par exemple, la liste `[1,2,1]` est une liste palindrome, alors que la liste `[1,2,1,3]` n'est pas une liste palindrome.
Ecrire une fonction `EstPalindrone(L)` prenant en argument une liste numérique `L` et renvoyant `True` si c'est une liste palindrome et `False` sinon.

Exercice 1.6 (Comptage) Ecrire une fonction `ComptageListe(L,val)` prenant en argument une liste numérique `L` et une valeur `val` et renvoyant le nombre de fois où `val` apparaît dans la liste `L`.
Par exemple :

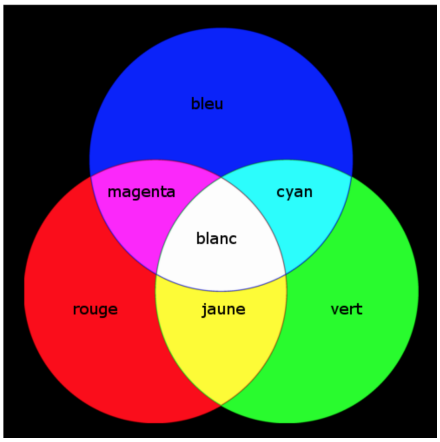
- `ComptageListe([1,1,5,5,1,4,5,3,5],5)` renvoie `4` ;
- `ComptageListe([1,1,5,5,1,4,5,3,5],10)` renvoie `0`.

2 Listes de listes et images numériques

Exercice 2.1 (Préliminaire : petit point sur le codage de la couleur)

On rappelle qu'une image couleur est composée de pixels comportant trois informations : les intensités de rouge, de vert et de bleu. Ces intensités sont codées par des valeurs entières comprises entre 0 et 255.

Le schéma ci-contre donne le principe du système additif de la couleur qui permet de visualiser comment la combinaison de plusieurs sources colorées lumineuses (du rouge, du vert et du bleu) produit les couleurs d'autres couleurs (ici le jaune, le magenta et le cyan).



Compléter le tableau ci-dessous avec le triplet de valeurs correspondant à la couleur.

Couleur	Noir	Rouge	Vert	Bleu	Cyan	Jaune	Magenta	Blanc
Triplet	$(0,0,0)$	$(255,0,0)$					$(255,0,255)$	

Exercice 2.2 (Création d'une image couleur pixel par pixel)

1. On donne le script Python ci-dessous.

```
def DessineImage() :
    img = [[0 for x in range(7)] for y in range(5)]
    for x in range(7) :
      img[0][x] = [255,0,0]
    for y in range(5) :
      img[y][6] = [0,0,255]
    for x in range(6) :
      for y in range(1,5) :
        if x == y :
          img[y][x] = [0,255,0]
        else :
          img[y][x] = [255,255,255]
```

Colorier le quadrillage représentant les pixels de l'image construite de la couleur qui convient :

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							

2. Quelles sont les couleurs utilisées dans cette image ?
3. Pourquoi le dernier pixel de la première ligne est-il bleu et pas rouge ?
4. Où sont situés les pixels verts ?
5. Comment modifier le code fourni afin que la deuxième ligne de l'image soit cyan, que la troisième colonne soit magenta et que la pixel à l'intersection de cette deuxième ligne et troisième colonne soit noir ?

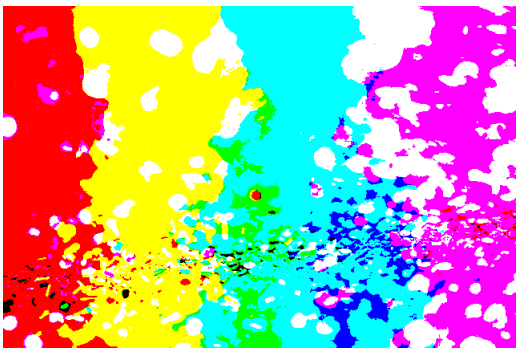
Exercice 2.3 (Filtres couleurs) On a donné la fonction suivante, qui, à partir d'une image couleur img codée sous la forme d'une liste de listes, produit une image bleue :

```
def FiltreBleu(img) :
    ''' Produit une image bleue à partir d'une image en
        couleur donnée sous forme matricielle (liste de lignes)
        en mettant les intensités de rouge et de vert à 0.
    '''
    hauteur = len(img) # nombre de lignes (hauteur de l'image)
    largeur = len(img[0]) # nombre de colonnes (largeur de l'image)
    new = [[0 for x in range(largeur)] for y in range(hauteur)]
    for y in range(hauteur) :
      for x in range(largeur) :
        new[y][x] = [0,0,img[y][x][2]]
    return new
```

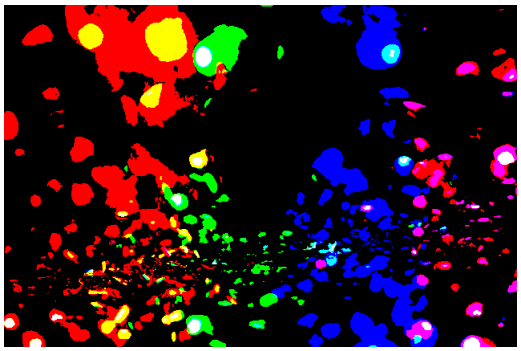
1. Comment modifier la fonction pour produire un filtre rouge ?
2. Même question pour un filtre vert.
3. Modifier le script afin d'échanger les quantités de rouge et de bleu.

Exercice 2.4 (Filtres couleurs secondaires) Pour modifier une image couleur on peut choisir d'appliquer un seuil sur chacune des intensités des pixels, comme on l'a fait sur les images en niveaux de gris pour produire une image noir et blanc. Les seules couleurs obtenue sont alors les 8 couleurs de l'exercice 1.
Voici ce que l'on pourrait obtenir en appliquant ce procédé sur une image en utilisant plusieurs valeurs de seuil (les trois valeurs sont dans l'ordre celle du seuil de rouge, de vert et de bleu) :

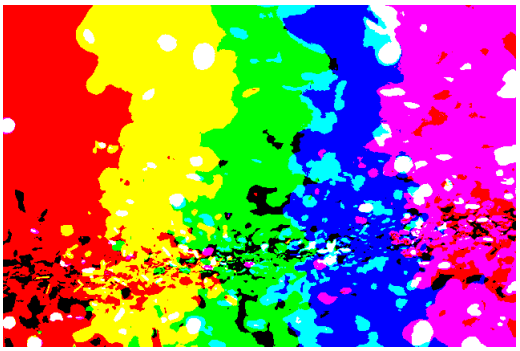
Image d'origine



Seuils : 50, 100, 200



Seuils : 200, 200, 200



Seuils : 100, 200, 50

On redonne ci-dessous la fonction `SeuilNB`, qui, à partir d'une image en niveaux de gris `img` codée sous la forme d'une liste de listes et d'une valeur de seuil comprise entre 0 et 255 produit une image en noir et blanc en effectuant un seuillage sur l'intensité de gris des pixels :

```
def SeuilNB(img, seuil) :
    hauteur = len(img) # nombre de lignes (hauteur de l'image)
    largeur = len(img[0]) # nombre de colonnes (largeur de l'image)
    new = [[0 for x in range(largeur)] for y in range(hauteur)]
    for y in range(hauteur) :
        for x in range(largeur) :
            if img[y][x] >= seuil :
                new[y][x] = 1 # le pixel est blanc
            else :
                new[y][x] = 0 # le pixel est noir
    return new
```

En vous inspirant de la fonction `SeuilNB` compléter le corps de la fonction `SeuilCouleur` suivante afin qu'elle produise ces effets sur l'image couleur passée en argument à partir de trois valeurs de seuils, nombres entiers compris entre 0 et 255, elles aussi passées en argument. Les intensités de chaque couleur de l'image obtenue seront égales à 0 ou à 255.

```
def SeuilCouleur(img, seuilR, seuilV, seuilB) :
    ''' Produit une image couleur modifiée à partir de la matrice
        img d'une image couleur de trois valeurs entières comprises entre 0 et 255.
        seuilR est le seuil pour le rouge, seuilV pour le vert, seuilB pour le bleu.
        Ces seuils indiquent quand les trois intensités des pixels valent 255 ou 0
    '''
```

Exercice 2.5 (Symétries) On donne le script d'une fonction nommée `SymétrieVerticale` :

```
def SymétrieVerticale(img) :
    ''' Produit une image par symétrie verticale à partir d'une
        image donnée sous forme matricielle (liste de lignes).
        Les pixels d'une même ligne sont inversés par rapport
        à l'image d'origine.
    '''
    hauteur = len(img) # nombre de lignes (hauteur de l'image)
    largeur = len(img[0]) # nombre de colonnes (largeur de l'image)
    new = [[0 for x in range(largeur)] for y in range(hauteur)]
    for y in range(hauteur) :
       for x in range(largeur) :
          new[y][x] = img[y][largeur-x-1]
    return new
```

A l'aide de cette fonction on a pu effectuer la transformation de symétrie verticale représentée ci-dessous à droite :

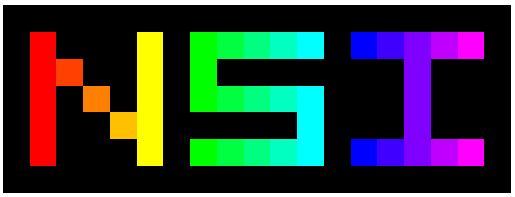
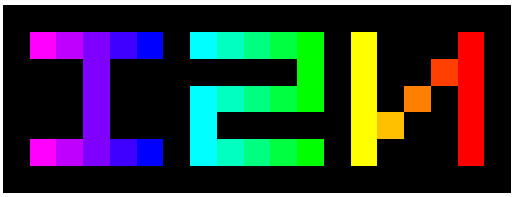


Image d'origine



Symétrie verticale

1. On donne la représentation sous forme de tableau d'une image composée de 8 pixels ci-dessous à gauche et la représentation incomplète de l'image produite par symétrie verticale à droite.

Image d'origine

y\x	0	1	2	3
0	(255,255,255)	(255,0,0)	(0,255,0)	(0,0,255)
1	(0,0,0)	(255,0,255)	(255,255,0)	(0,255,255)

Image symétrique

y\x	0	1	2	3
0	(0,0,255)			
1			(255,0,255)	

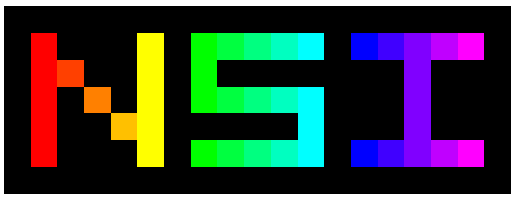
   (a) Compléter le tableau de droite avec les couleurs qui conviennent pour compléter l'image symétrique.

   (b) Dans l'image symétrique le pixel bleu, ligne 0 colonne 0, est le symétrique du pixel ligne 0 colonne 3 dans l'image d'origine. De la même façon, le pixel magenta, ligne 1 colonne 2 dans l'image symétrique correspond au pixel ligne 1 colonne 1 dans l'image d'origine.
   Quelle est la position du pixel noir dans l'image symétrique ? Et celle du pixel vert ?

   (c) Des pixels symétriques sont-ils situés sur la même colonne ? La même ligne ?

   Pour placer le pixel symétrique dans l'image de droite, on conserve ainsi la même ligne (la valeur de `y`) mais on modifie la colonne en calculant `largeur-x-1`. Le `-1` prend en compte qu'en informatique les indices commencent à 0.

2. Ecrire une fonction `SymétrieHorizontale` pouvant produire l'image ci-contre en bas à partir de l'image NSI ci-contre en haut :



Symétrie horizontale