

Pour chaque exercice des exemples de code, structures de données et tests sont fournis en annexe.

## Exercice 1 : Charger une image et dessiner (1h10min)

Une image en informatique est stockée sous forme d'une matrice de pixels qui contiennent les couleurs. Chaque couleur est encodée sous forme de triplets (r,g,b) (Red, Green, Blue). Voici un exemple de code permettant de charger une image sous ce format en Python (en utilisant la bibliothèque Pillow dont le module Python s'appelle PIL) et initialiser une image identique vide :

```
from PIL import Image

im = Image.open("lyon.png")
px = im.load()

W, H = im.size          # taille de l'image
r, g, b = px[x, y]      # récupère un pixel
px[x, y] = r, g, b      # change un pixel

im.show()               # affiche l'image

# créer une image identique vide (noir)
im2 = Image.new('RGB', (im.width, im.height))
px2 = im2.load()
```



**Exercice 1.1 (10min)** – Définissez une fonction d'écriture d'un pixel à une position (x,y) d'une image avec une couleur donnée en argument; de même une fonction de lecture qui renvoie la couleur du pixel (x,y).

**Exercice 1.2 (10min)** – Écrire une fonction permettant de peindre un rectangle de l'image avec une même couleur. Coloriez avec la couleur moyenne de cette région (et donc définir une fonction qui la calcule).

**Exercice 1.3 (10min)** – Ecrire une fonction de calcul de distance Euclidienne entre deux couleurs. Attention il s'agit de couleurs à comparer et non pas des positions.

$$d(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2} \quad (1)$$

**Exercice 1.4 (40min)** Proposez une méthode de remplissage d'une image en régions homogènes (via un critère tel que l'écart-type) en utilisant l'algorithme dit de *flood filling* et qui fonctionne comme suit :

1. Charge une image et mettre les pixels dans une liste
2. Définir un point de départ
3. Explorer les 4 directions :
  - (a) Si la valeur est au dessus d'un seuil d'homogénéité alors on ne remplit pas
  - (b) Si la valeur est en dessous alors on remplit et on ajoute les voisins dans la liste
  - (c) Répéter cela pour les pixels non-explorés dans l'image

Conseil : vous pouvez utiliser la distance Euclidienne comme critère d'homogénéité entre la couleur de départ et les couleurs des voisins (à des fins de test, vous pouvez utiliser 200 comme seuil de distance maximale de couleur avec le pixel (0,0)).

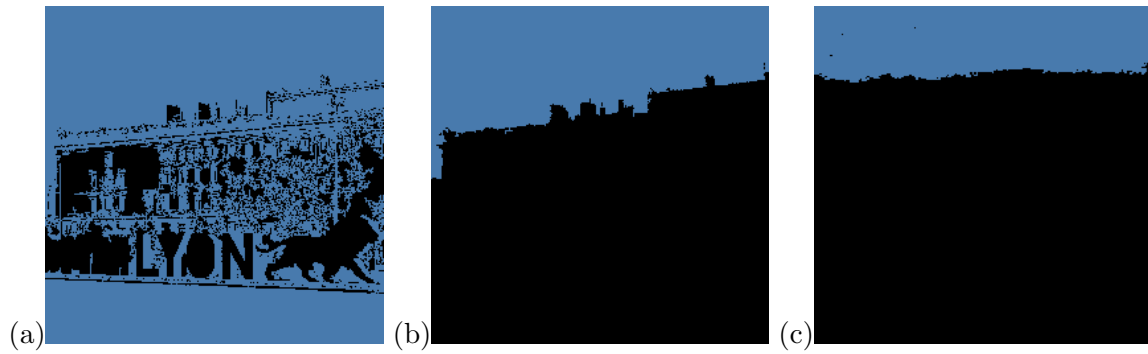


Figure 1: (a) seuil 200 (b) seuil 100 (c) seuil 20

## Exercice 2 : Traitement d'image (50min)

Le traitement d'image permet d'extraire des informations intéressantes d'une image (contours, formes, etc.). Le flou gaussien est une étape importante de ce traitement afin de réduire le bruit que les images peuvent contenir. Il utilise une opération dite de *convolution*, permettant d'appliquer une fonction de distribution gaussienne aux voisins d'un pixel et d'en faire la moyenne. Autrement dit il s'agira de réaliser la moyenne pondérée de chaque pixel en réalisant la moyenne du pixel et de ses voisins en utilisant par exemple la matrice ci-dessous (dont les valeurs sont définies par la distribution gaussienne donnée en annexe pour une matrice  $3 \times 3$ ) :

$(x-1,y-1)$	$(x,y-1)$	$(x+1,y-1)$
$(x-1,y)$	$(x,y)$	$(x+1,y)$
$(x-1,y+1)$	$(x,y+1)$	$(x+1,y+1)$

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

**Exercice 2. (5min)** - Écrire une fonction de conversion d'image en niveaux de gris (soit la moyenne des triplets  $(r, b, g)$  ou en utilisant la formule suivante :  $((0.3 \times R) + (0.59 \times G) + (0.11 \times B))$ ).

**Exercice 2.2 (30min)** - Implémentez le filtre gaussien tel que défini ci-dessus en définissant une méthode *convolution* qui effectue la multiplication des pixels centrés sur le pixel en cours avec les valeurs de la matrice, puis la somme pondérée. Vous pourrez utiliser les matrices en annexe d'approximation du filtre (en commençant par le filtre Gaussien  $3 \times 3$  dont le total des valeurs est 16). Utilisez la version de l'image en niveaux de gris afin de simplifier les traitements.

**Exercice 2.3 (15min)** - Généralisez votre programme pour tout type de filtre (en taille et en valeur).

**Exercice 2.3 (Bonus)** - Calculez et affichez la différence entre l'image originale et l'image filtrée afin de mettre en avant les changements que le filtre a apporté.

## Annexe : Exemples de filtres

```
gauss3 = [[1,2,1],
          [2,4,2],
          [1,2,1]]

gauss7 = [[1,1,2,2,2,1,1],
          [1,2,2,4,2,2,1],
          [2,2,4,8,4,2,2],
          [2,4,8,16,8,4,2],
          [2,2,4,8,4,2,2],
          [1,2,2,4,2,2,1],
          [1,1,2,2,2,1,1]]

sobely3 = [[-1, 0, 1],
           [-2, 0, 2],
           [-1, 0, 1]]

sobelx3 = [[-1, -2, -1],
           [0, 0, 0],
           [1, 2, 1]]
```

Exemple de résultat de flou gaussien en utilisant PIL :

```
_im = im.filter(ImageFilter.GaussianBlur)
_im.show()
```

Résultat :

