À rendre par mail avant le 15 janvier 2018 :
alexandre.benoit@ac-versailles.fr

Le but de ce DM est de programmer quelques fonctionnalités du logiciel **Mesurin**. On complétera le fichier python : **dm_1.py** disponible sur le site web. Le fichier devra juste contenir les fonctions demandées. Comme lors du TD, on ne travaillera pas sur l'image directement mais sur le tableau contenant la valeur des pixels de l'image.

On testera ces programmes à partir des deux images couleurs **retine.jpg** et **stomates.jpg** (ces images devront se trouver dans le même dossier que le fichier python).

On rappelle que le tableau d'une image couleur est semblable au tableau d'une image niveau de gris, sauf que chaque Pixel est représenté par un triplet de nombres entre 0 et 255 au lieu d'un nombre.

Par exemple : `image_tab[i,j][1]` est un nombre entre 0 et 255 qui représente l'intensité du vert (le 0 est le rouge, le 1 le vert et le 2 le bleu).

Pour créer une image couleur remplie de 0, il faut initialiser les triplets de couleur. On la déclarera comme :

```
image_tab = numpy.zeros([nb_lignes, nb_colonnes, 3], dtype=numpy.uint8)
```

Attention à bien configurer Pyzo comme indiqué sur le site.
En cas de difficultés de configuration, contacter moi par mail en précisant exactement le problème (ce qui a été fait et ce qui ne marche pas).

Exercice 1 : Décomposition d'une image couleur en image de niveau de gris

La décomposition d'une image consiste à ne garder qu'une seule des trois couleurs.

(1) Écrire la fonction **image_rouge_gris** qui partant d'une image en couleur renvoie une image en niveau de gris dont chaque pixel contient l'intensité en rouge de l'image précédente. Par exemple si un pixel de l'image d'origine a comme intensité $(142, 12, 253)$, la valeur du même pixel de l'image de niveau de gris sera 142.

(2) Écrire la fonction **image_rouge** qui partant d'une image en couleur renvoie une image en couleur dont chaque pixel ne contient que l'intensité en rouge de l'image précédente. Par exemple si un pixel de l'image d'origine a comme intensité $(142, 12, 253)$, la valeur du même pixel de l'image en couleur sera $(142, 0, 0)$.

Exercice 2 : Rotation d'une image

Rappel : Une rotation de 90 *degré consiste à tourner l'image d'un quart de tour vers la gauche*

Écrire la fonction **rotation** qui effectue la rotation de 90 degré de l'image.

Attention pour cette fonction, les dimensions de l'image peuvent changer. En effet si l'image fait 200×50 *pixels, avec la rotation elle fera* 50×200 *pixels. On inversera donc bien la hauteur et l'image dans la déclaration du nouveau tableau.*

Exercice 3 : Redimensionner une image

Encore une fois, les dimensions des images vont changer.

(1) Écrire la fonction **agrandissement** qui double la taille d'une image (quadruple le nombre de pixels). Cette fonction créera une image contenant deux fois plus de ligne et deux fois plus de colonnes. Elle remplacera chaque pixel par un bloc de 2×2 pixels.

(2) Écrire la fonction **reduction** qui divise par deux la taille d'une image. Chaque bloc de 2×2 pixels sera remplacé par un unique pixel dont la valeur est la moyenne des 4 pixels originaux.

Exercice 4 : Créer un coloriage

Dans cet exercice, on considère une image noir et blanc comme une image de niveau de gris avec 2 valeurs possibles par pixel : 0 et 255.

On utilisera les fonctions du TP sur les images si nécessaire.

(1) Écrire une fonction **contour_couleur** qui prend en argument une image couleur et
   a. appelle une fonction pour transformer l'image couleur en image de niveau de gris puis
   b. appelle une fonction pour obtenir les contours de l'image.

   Cette fonction renvoie le contour de l'image.

Les contours ne sont pas tous fermé. On souhaite donc maintenant fermer ces contours.

(2) Écrire la fonction **dilatation** qui prend en argument une image noir et blanc et qui rend noir un pixel si au moins un de ses voisins est noir et le rend blanc sinon.

(3) Écrire la fonction **erosion** qui prend en argument une image noir et blanc et qui rend noir un pixel si tous ses voisins sont noirs et le rend blanc sinon.

(4) Écrire la fonction fermeture qui prend en argument une image noie et blanc et applique la fonction **dilatation** puis **erosion** sur l'image.

(5) En déduire une fonction **coloriage**.

Plutôt que de ne considérer que les 8 voisins, on peut se donner un paramètre r et faire le test sur tous les pixels de coordonnées d'abscisses comprises entre $[\![i-r; i+r]\!]$ et d'ordonnées comprises entre $[\![j-r; j+r]\!]$