I Description d'une image.

Les images sont manipulées en *Python* sous la forme de tableaux de *pixels*, chacun contenant des informations de couleur.

En mode *RGB*, pour "Red-Green-Blue", chaque pixel est un **triplet d'entiers** compris entre 0 et 255. La première coordonnée du pixel indique le niveau de rouge, la seconde de vert, et la dernière, de bleu.

Si toutes les valeurs sont nulles, le pixel est noir. Si toutes sont maximales (255), le pixel est blanc.

Si toutes sont égales à une valeur g, le pixel est gris.
On parle alors d'un *niveau de gris* (g proche de 0 : gris foncé, g proche de 255 : gris clair).

Cette combinaison RGB autorise 16 777 216 couleurs possibles (256^3)

Dans une image, chaque pixel est repéré par ses coordonnées (i, j) : i repère la colonne à laquelle appartient le pixel, et j repère la ligne.

Ainsi, le pixel en haut à gauche d'une image rectangulaire est de coordonnées $(0, 0)$, alors que le pixel en bas à droite est de coordonnées $(n-1, p-1)$ où n désigne le nombre de colonnes de l'image, et p désigne son nombre de lignes.

On dira d'une telle image qu'elle est de *taille* $n \times p$. Elle contient $n \times p$ pixels.

Attention ! Contrairement à l'usage pour les matrices, la première coordonnée désigne l'indice de colonne, et la deuxième, de ligne.

II Commandes pour démarrer

Répertoire de travail : Avec *Pyzo*, une fenêtre appelée **File browser** (qu'on peut choisir d'afficher ou pas par l'intermédiaire de l'onglet **Tools**) indique quel est le répertoire de travail, où toutes vos images seront enregistrées. Pour définir le répertoire de travail, utiliser la barre d'adresse du *File browser* pour sélectionner le répertoire désiré, puis cliquer sur l'étoile et choisir *Go to this directory in the current shell.*

Module de traitement d'image : `from PIL import Image as im`

Les commandes utilisées par la suite seront préfixées du nom de ce module : `im.commande`.

Travailler à partir d'une image existante : `A = im.open("Info04-orni.jpg")`

Ceci crée une variable, modifiable, à partir d'une image préexistante du répertoire de travail.

Enregistrer une nouvelle image : `A.save("ornimodif.jpg")`

Ceci enregistre dans le répertoire de travail, au format *jpg*, l'image contenue dans la variable.

Afficher une image : `A.show()`

Créer une image : `A = im.new("RGB",(n,p),(255,255,255))`

Ceci crée une image sous le nom `A`, en mode *RGB*, de taille $n \times p$, et dont chaque pixel contient la couleur $(255, 255, 255)$, c'est-à-dire est blanc (à changer si besoin).

Copier une image : `B = A`

On utilise l'affectation de variable pour créer une nouvelle variable, qui est une copie de `A`.

III Commandes de traitement d'images

Obtenir la taille d'une image : `A.size`

Renvoie le couple (n, p) définissant la taille de l'image `A`. Il s'agit d'un `tuple` dont on peut extraire les coordonnées en utilisant `A.size[0]` ou `A.size[1]`.
On peut aussi définir directement les entiers donnant les dimensions d'une image : `(n,p) = A.size`

Obtenir un pixel d'une image : `A.getpixel((i,j))`

Cette commande renvoie le triplet correspondant au pixel de position (i, j).
On peut en extraire :

- la valeur de rouge : `A.getpixel((i,j))[0]`
- la valeur de vert : `A.getpixel((i,j))[1]`
- la valeur de bleu : `A.getpixel((i,j))[2]`

Il est aussi commode de sauvegarder ce `tuple` comme variable : `pixel = A.getpixel((i,j))`
et l'on pourra ensuite accéder à la couleur désirée en appellant `pixel[`k`]`, pour $0 \leqslant k \leqslant 2$.
On peut également utiliser la syntaxe plus simple : `(r,g,b) = A.getpixel((i,j))`
et alors la variable `r` sera égale à la valeur de rouge, `g` à celle de vert et `b` à celle de bleu.

Modifier un pixel d'une image : `A.putpixel((i,j),(`r`, `g`, `b`))`

où r, g et b sont des **entiers** entre 0 et 255, et où (i, j) est la position du pixel qu'on modifie.

Modifier la taille d'une image : `A.resize((n2,p2))`

Crée une nouvelle image en étirant ou contractant l'image d'origine pour lui donner la taille (n_2, p_2).

Faire une *fondue* de deux images : `im.blend(A1,A2,`α` )`

Crée une nouvelle image à partir des deux images de même taille appelées en arguments, en faisant pour chaque pixel une moyenne *barycentrique* des couleurs de chaque image d'origine, et en utilisant le coefficient $0 \leqslant \alpha \leqslant 1$ désignant le poids de la seconde image ($1 - \alpha$ étant le poids de la première).

IV Travail à réaliser

1 Un spectre de couleurs

- Créer une nouvelle image de taille 256×256, monochrome rouge, puis l'afficher.
- Modifier cette image de sorte que la première colonne reste rouge, la dernière soit bleue, et que chaque colonne entre ces deux extrêmes présente un dégradé régulier du rouge vers le bleu.
  Afficher votre image (libre à vous de l'enregistrer dans votre répertoire de travail).

2 Convertir une image en gris

- Sélectionnez une image disponible dans votre répertoire de travail, et l'importer avec *Python*.
- Modifier cette image de sorte que chaque pixel soit gris : le niveau de gris sera la moyenne des 3 valeurs R,G,B.
- Afficher votre image (libre à vous de l'enregistrer dans votre dossier personnel).
- Créer une fonction `gris` qui prend comme argument une image, et renvoie une image grisée de la façon ci-dessus.

3 Convertir une image en noir et blanc

Créer une fonction `noirBlanc` qui prend comme arguments une image et un seuil $k \in [\![0, 255]\!]$ et qui renvoie une image ne contenant que des pixels ou bien noirs $(0, 0, 0)$, ou bien blancs $(255, 255, 255)$ selon que la moyenne des couleurs R, G, B est supérieure ou inférieure au seuil choisi k.

4 Obtenir le négatif d'une image

Créer une fonction `negatif` qui prend comme argument une image et renvoie l'image *complémentaire*, c'est-à-dire celle dont les valeurs de couleurs de chaque pixel complètent les valeurs initiales à 255.

5 Obtenir la symétrie d'une image

Créer une fonction `miroir` qui prend comme argument une image et renvoie l'image *symétrique*, selon un axe vertical passant par le centre de l'image.

6 Pivoter une image

Créer une fonction `pivoter` prenant pour arguments une image et un argument n entier tel que $1 \leqslant n \leqslant 3$, et qui renvoie l'image pivotée de n quarts de tour dans le sens trigonométrique.

7 Flouter une image

Créer une fonction `flou` renvoyant une image floutée de la façon suivante : chaque pixel reçoit les valeurs RGB moyennes du pixel d'origine et de tous ses voisins.
Que se passe-t-il lorsqu'on applique un grand nombre de fois cette fonction à une image ?

8 Augmenter le contraste d'une image

Créer une fonction `contraste` qui prend comme arguments une image et une valeur $c \in [0, 127]$ et qui renvoie une version plus contrastée de l'image d'origine, selon un paramètre c.
On peut par exemple, pour chaque pixel de l'image et chaque composante RGB :

- Si la valeur de couleur est inférieure à c, lui donner la valeur 0,
- si la valeur de couleur est supérieure à $255 - c$, lui donner la valeur 255,
- et sinon effectuer une transformation affine (et continue) sur cette valeur.

9 Mesurer les niveaux moyens de couleurs *R, G, B*

Créer des fonctions `rouge, vert, bleu` qui prennent comme argument une image et renvoient la moyenne des composantes R,G,B de ses pixels.
Ces fonctions pourront être utilisées pour "mesurer" la présence d'une couleur dans une image.
Il pourra être important de rapporter le résultat au niveau d'éclairage de l'image.

10 Dessiner le contour d'une image

Créer une fonction `contour` qui prend comme arguments une image et une valeur s et qui renvoie une image contenant les contours de l'image d'origine, c'est-à-dire les pixels pour lesquelles les valeurs des couleurs sont significativement différentes des valeurs voisines.
Cette différence sera appréciée par un seuil s.

Il y a ici plusieurs façons de procéder, qui ne donnent pas forcément les mêmes résultats.
Soyez imaginatifs !

V Pour aller plus loin

Les exercices qui suivent sont hors-programme.

1 Chasser le gris

Créer une fonction `psyche` qui prend comme arguments une image et une valeur s et qui renvoie une image dans laquelle tous les pixels mesurés comme *gris* selon le seuil s auront leurs couleurs modifiées pour s'éloigner du gris.

Pour mesurer si un pixel est gris, on peut calculer la moyenne m des valeurs R, G, B du pixel, puis calculer une "distance" du pixel à m :

- $d_1 = |R - m| + |G - m| + |B - m|$,
- $d_2 = \sqrt{(R - m)^2 + (G - m)^2 + (B - m)^2}$,
- $d_3 = \max(|R - m|, |G - m|, |B - m|)$,
- ...

Pour *éloigner* un pixel du gris, on peut modifier arbitrairement l'une (ou plusieurs) de ses composantes selon une fonction à choisir.

2 Façon Andy Warhol

Créer une fonction `warhol` qui prend pour arguments une image, un multiplicateur m et un seuil s et qui crée une image composée de $m \times m$ fois l'image d'origine, disposées en carré, et dont chacune a été modifiée par la fonction `psyche` selon le paramètre $k \times s$ pour $0 \leqslant k \leqslant m^2 - 1$.

Attention à l'utilisation de cette fonction sur de grandes images, le temps de calcul peut s'allonger...
On peut, si besoin, réduire préalablement la taille de l'image d'origine.