Python – Manipulation d'images

Le module Pillow

Python offre la possibilité de créer et de modifier à l'aide de différents modules. Nous allons utiliser le module Pillow. Pour travailler sur les images, nous allons manipuler des tableaux à 2 dimensions de taille hauteur×largeur. Ainsi `tab[0][0]` correspondra au coin supérieur gauche, `tab[0][largeur-1]` au coins supérieur droit, `tab[hauteur-1][0]` au coin inférieur gauche et `tab[hauteur-1][largeur-1]` au coin inférieur droit.
Les cases du tableau seront remplis par des triplets `(R, G, B)`, où chaque valeur est un entier entre 0 et 255. `R` est la quantité de rouge, `G` est la quantité de vert et `B` est la quantité de bleu. On peut aussi rajouter une quatrième valeur, `A`, qui correspond à la transparence.
Le début de votre fichier doit contenir :

```
from PIL import Image

# Couleurs de base
BLANC = (255, 255, 255)
NOIR = (0, 0, 0)
ROUGE = (255, 0, 0)
VERT = (0, 255, 0)
BLEU = (0, 0, 255)

def creation_matrice(largeur, hauteur, fond=BLANC):
    return [[fond]*largeur for _ in range(hauteur)]
```

Vous pouvez rajouter d'autres couleurs si vous le souhaitez. Conformément aux conventions de nommage Python, les constantes sont nommées en majuscule.
La fonction `creation_matrice` permet de créer un tableau à 2 dimensions rempli, par défaut, avec des pixels blancs ou de la couleur donnée en paramètre.

Préparation

Pillow utilise en interne un tableau à 1 dimension de pixels pour représenter l'image ligne par ligne, en allant de gauche à droite et de bas en haut. Pour que nous puissions utiliser des tableaux à 2 dimensions, il faut pouvoir les convertir en 1 dimension et réciproquement.

Exercice 1 : Compléter le code de la fonction `conv_2d_1d(tab)` qui prend un tableau à 2 dimensions et qui retourne un tableau à 1 dimension qui correspond aux lignes de `tab` mis bout à bout.

```
def conv_2d_1d(tab):
    m = len(tab)
    n = len(tab[0])
    tab1 = [0] * ...
    for i in range(m):
        for j in range(n):
            tab1[...] = tab[i][j]
    return tab1
```

```
>>> conv_2d_1d([[1, 2, 3], [4, 5, 6]])
[1, 2, 3, 4, 5, 6]
```

Exercice 2 : Compléter le code de la fonction `conv_1d_2d(tab, m, n)` qui prend un tableau à 1 dimension et qui retourne un tableau à 2 dimensions qui correspond à `tab` découpé en `m` lignes de longueur `n`.

```
def conv_1d_2d(tab, m, n):
    tab2 = [[0]*n for _ in range(m)]
    for i in range(m):
        for j in range(n):
            tab2[i][j] = tab[...]
    return tab2
```

```
>>> conv_1d_2d([1, 2, 3, 4, 5, 6, 7, 8], 4, 2)
[[1, 2], [3, 4], [5, 6], [7, 8]]
>>> conv_1d_2d([1, 2, 3, 4, 5, 6, 7, 8], 2, 4)
[[1, 2, 3, 4], [5, 6, 7, 8]]
```

Nous pouvons maintenant définir les fonctions qui serviront à importer une image et à afficher une image à partir d'une matrice :

```
def image_vers_matrice(nom):
    image = Image.open(nom)        # on ouvre le fichier
    tab = list(image.getdata())    # on recupere le tableau 1d des pixels
    largeur, hauteur = image.size # on recupere les dimensions
    return conv_1d_2d(tab, hauteur, largeur) # on renvoie le tableau 2d

def affiche_image(matrice, nom_export=""):
    largeur = len(matrice[0])
    hauteur = len(matrice)
    image = Image.new('RGB', (largeur, hauteur)) # creation de l'image
    image.putdata(conv_2d_1d(matrice))           # transfert des pixels
    if nom_export != "":          # on regarde si on doit sauver le fichier
        image.save(nom_export)
    image.show()           # on affiche
```

Vous pouvez choisir une image, par exemple `mongolfiere.jpg`, la copier dans le dossier où se trouve le fichier Python et tester :

```
>>> affiche_image(image_vers_matrice("mongolfiere.jpg"))
```

Si vous souhaitez enregister l'image obtenue, ce qui sera plus intéressant par la suite, vous pouvez rajouter un nom de fichier à la fonction `affiche_image`.

```
>>> affiche_image(image_vers_matrice("mongolfiere.jpg"), "mongolfiere2.jpg")
```

Manipulation de base d'une image

Il es possible d'extraire une couche correspondant à une des composantes de la couleur. Par exemple, la fonction suivante permet de ne garder que les valeurs de rouge dans l'image.

```
def couche_rouge(nom, nom_export=""):
    matrice_originale = image_vers_matrice(nom)   # on recupere l'image
    largeur = len(matrice_originale[0])   # on recupere les dimensions
    hauteur = len(matrice_originale)
    matrice = creation_matrice(largeur, hauteur) # on cree la matrice
    for ligne in range(hauteur):
        for col in range(largeur):
            r, g, b = matrice_originale[ligne][col]
            matrice[ligne][col] = (r, 0, 0)   # on ne garde que le rouge
    affiche_image(matrice, nom_export) # on affiche
```

```
>>> couche_rouge("mongolfiere.jpg")
```

Pour l'instant, nous allons afficher directement l'image obtenue, mais il serait possible de renvoyer la matrice pour effectuer d'autres opérations dessus. De même la fonction pourrait prendre une matrice en paramètre et travailler directement dessus.

Exercice 3 : Écrire les fonctions permettant d'obtenir la couche verte et la couche bleue.

Exercice 4 : Écrire une fonction `negatif(nom, nom_export="")` qui affiche le négatif de l'image. Le négatif est obtenu en remplaçant les composantes R, G et B par 255 – R, 255 – G et 255 – B.

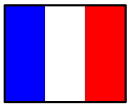
Exercice 5 : Écrire une fonction `miroir_horizontal(nom, nom_export="")` qui affiche l'image où la gauche et la droite ont été inversés.

Exercice 6 : Écrire une fonction `miroir_vertical(nom, nom_export="")` qui affiche l'image où le haut et le bas ont été inversés.

Création d'images

Dans cette partie, nous allons générer directement des images. Tous les exercices suivants peuvent être réalisés en utilisant une seule boucle imbriquée, comme dans les fonctions précédentes. Pour simplifier les tests, il est conseillé de mettre des valeurs par défaut à toutes les variables. Vous pouvez également rajouter le paramètre optionnel `nom=""` qui sera passé à `affiche_image` pour sauvegarder l'image si vous le souhaitez.

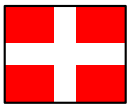
Exercice 7 : Compléter le code de la fonction `drapeau_tricolore(largeur, hauteur, coul1, coul2, coul3)` qui affiche un drapeau tricolore, dont les dimensions sont `largeur` et `hauteur`, composé de 3 bandes verticales de même largeur.



```
>>> drapeau_tricolore(400, 300, BLEU, BLANC, ROUGE)
```

```
def drapeau_tricolore(largeur, hauteur, couleur1, couleur2, couleur3, nom=""):
    matrice = creation_matrice(largeur, hauteur)
    for i in range(hauteur):
        for j in range(largeur):
            if ...:
                matrice[i][j] = couleur1
            elif ...:
                matrice[i][j] = couleur2
            else:
                matrice[i][j] = couleur3
    affiche_image(matrice, nom)
```

Exercice 8 : Écrire une fonction `croix(largeur, hauteur, epaisseur, coul1, coul2)` qui affiche une image dont le fond est `coul1` et qui a une croix de couleur `coul2` centrée au milieu de l'image et dont l'épaisseur en largeur et en hauteur est `epaisseur`.



```
>>> croix(400, 300, 50, ROUGE, BLANC)
```

Exercice 9 : Écrire une fonction `croix_suisse(largeur, hauteur, epaisseur, marge, coul1, coul2)` qui affiche une image dont le fond est `coul1` et qui a une croix de couleur `coul2` centrée au milieu de l'image et dont l'épaisseur en largeur et en hauteur est `epaisseur`. La distance entre la croix et chacun des bords de l'image est `marge`.

```
>>> croix_suisse(400, 300, 50, 20, ROUGE, BLANC)
```

Exercice 10 : Compléter le code de la fonction `decoupe_diagonale1(cote, coul1, coul2)` qui affiche une image carrée dont les deux côtés mesurent `cote` et coupée en 2 selon la diagonale allant du coin en haut à gauche à celui en bas à droite.

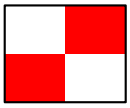
```
>>> decoupe_diagonale1(300, BLANC, ROUGE)
```

```
def decoupe_diagonale1(cote=200, couleur1=BLANC, couleur2=NOIR, nom=""):
    matrice = creation_matrice(cote, cote, couleur1) # on remplit couleur1
    for i in range(cote):
        for j in range(cote):
            if ...:
                matrice[i][j] = couleur2
    affiche_image(matrice, nom)
```

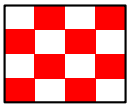
Exercice 11 : Écrire une fonction `decoupe_diagonale2(cote, coul1, coul2)` qui affiche une image carrée dont les deux côtés mesurent `cote` et coupée en 2 selon la diagonale allant du coin en haut à droite à celui en bas à gauche.

Exercice 12 : Écrire une fonction `decoupe_diagonale3(cote, coul1, coul2)` qui affiche une image carrée dont les deux côtés mesurent `cote` et coupée en 4 selon les diagonales.

Exercice 13 : Écrire une fonction `quadrants(largeur, hauteur, coul1, coul2)` qui affiche une image découpée en quatre rectangles.

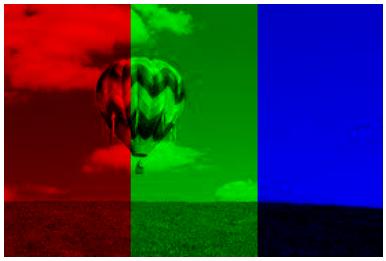


Exercice 14 : Écrire une fonction `damier(largeur, hauteur, n, coul1, coul2)` qui affiche une image découpée `n` rectangles sur la largeur et `n` rectangles sur la longueur.



Pour aller plus loin

Exercice 15 : Écrire une fonction `couches_tricolores(nom)` qui permet d'afficher un images où la partie gauche est la couche de rouge, celle du milieu est celle du vert et celle de gauche est la couche du rouge. Par exemple, pour avec l'image de la mongolfière, on obtient l'image ci-contre.



Exercice 16 : Compléter la fonction ci-dessous afin d'obtenir un montage des 3 couches ainsi que de l'image. Les dimensions de l'image produite sont le double de celles de l'originale.

```
def quadricolore(nom, nom_export=""):
    matrice_originale = image_vers_matrice(nom)   # on recupere l'image
    largeur = len(matrice_originale[0])   # on recupere les dimensions
    hauteur = len(matrice_originale)
    matrice = creation_matrice(..., ...) # on cree la matrice
    for ligne in range(hauteur):
        for col in range(largeur):
            r, g, b = matrice_originale[ligne][col]
            matrice[...][...] = (r, 0, 0)   # en haut à gauche
            matrice[...][...] = (0, g, 0)   # en haut à droite
            matrice[...][...] = (0, 0, b)   # en bas à gauche
            matrice[...][...] = (r, g, b)   # en bas à droite
    affiche_image(matrice, nom_export) # on affiche
```