Python – Rotation d'une image

Manipulation d'une image

Pour manipuler les images, nous allons utiliser le module `pillow` de Python. Pour cela, il faut mettre la ligne suivante au début de votre fichier :

```
from PIL import Image
```

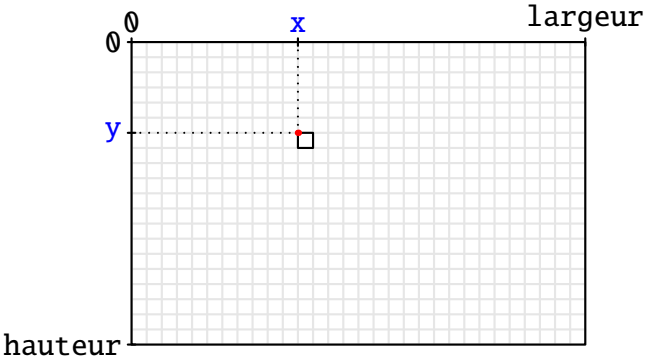
Vous pouvez utiliser l'image `fractale.jpg` qui se trouve dans `Echange` pour tester vos fonctions.
Pour charger une image et l'afficher, il faut utiliser les lignes suivantes :

```
>>> image = Image.open("fractale.jpg")
>>> image.show()
```

Pour avoir les dimensions de l'image et avoir accès aux pixels, il faut faire :

```
>>> largeur, hauteur = image.size
>>> pixels = image.load()
```

L'objet `pixels` fonctionne comme un tableau à deux dimensions contenant chacun des pixels.
Les coordonnées fonctionnent de la manière suivante :



Les coordonnées du pixel en bas à droite sont `(largeur-1, hauteur-1)`.
Pour obtenir la valeur d'un pixel ou la modifier, il faut faire :

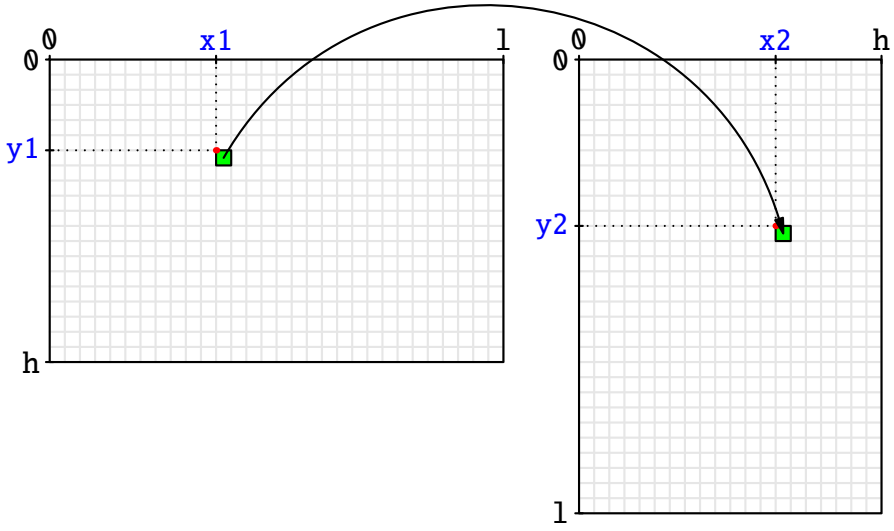
```
>>> couleur = pixels[5, 2]   # x=5 et y=2
>>> pixels[5, 2] = couleur
```

Pour créer et sauvegarder une image, il faut utiliser les commandes :

```
>>> image2 = Image.new("RGBA", (hauteur, largeur))
>>> image2.save("bonjour.png")
```

Rotation naïve

Pour faire une rotation d'un quart de tour, on peut créer une nouvelle image en inversant la hauteur et la largeur et ensuite copier tous les pixels.



Exercice 1 (sur papier) : Déterminer les coordonnées `(x2, y2)` du pixel correspondant à `(x1, y1)` de l'image d'origine.

Exercice 2 : Compléter le code de la fonction suivante effectuant cette rotation.

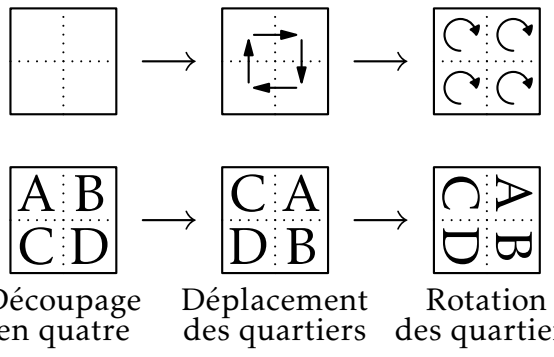
```
def rotation_naive(nom):
    image = Image.open(nom)
    largeur, hauteur = image.size
    pixels = image.load()
    image2 = Image.new("RGBA", (hauteur, largeur))
    pixels2 = image2.load()
    for x in range(largeur):
        for y in range(hauteur):
            pixels2[..., ...] = pixels[x,y]
    image2.show()
```

Le problème de cette méthode, c'est qu'elle double la mémoire utilisée à cause de la duppli-cation de l'image, mais elle marche pour toute taille d'image.

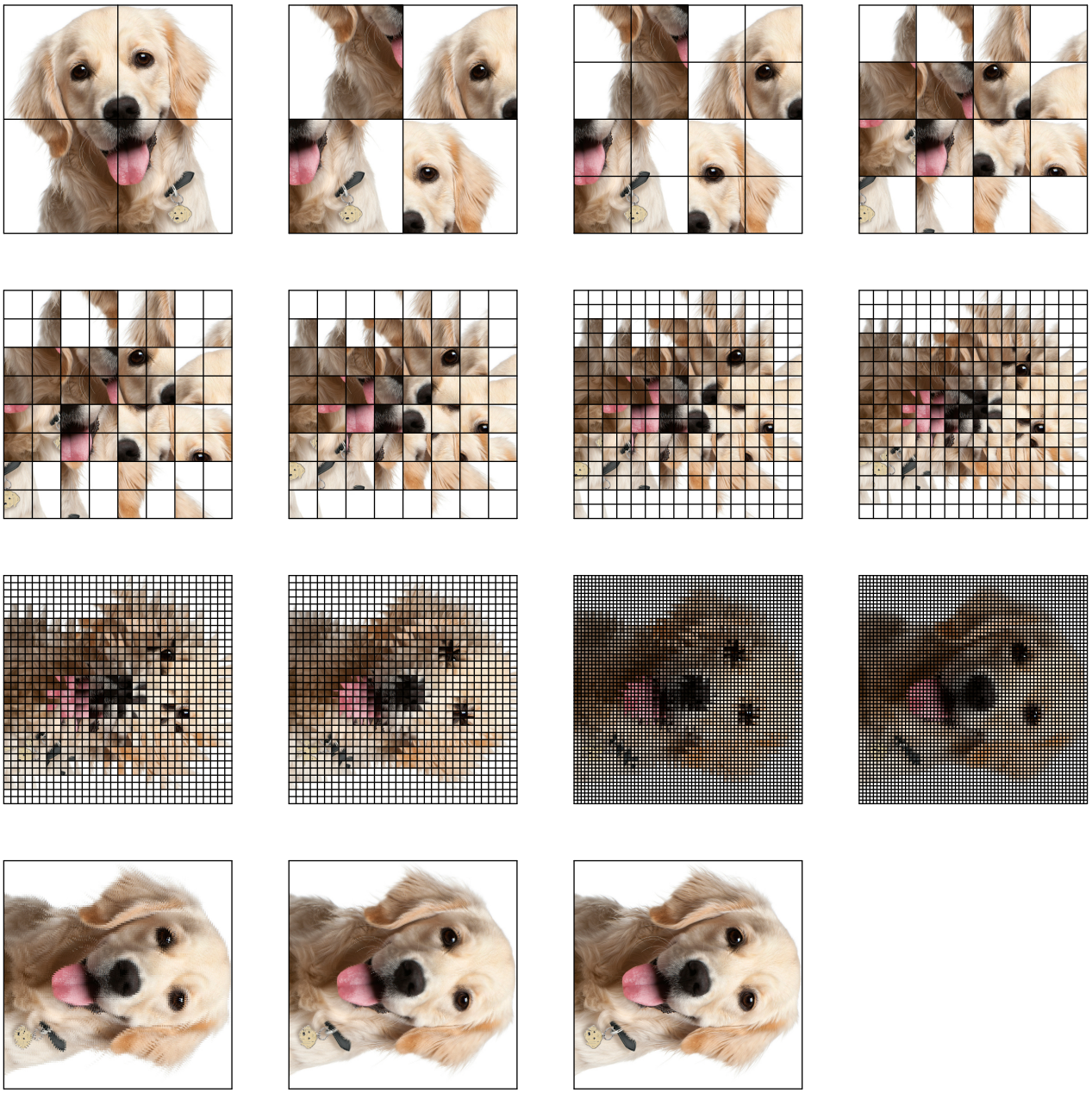
Diviser pour tourner

Si l'image est carrée et que la largeur est une puissance de 2, on peut utiliser la méthode "diviser pour régner" qui fonctionne de la manière suivante :

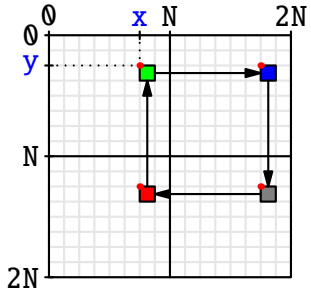
- On coupe l'image en 4 quartiers.
- On échange la position des quartiers.
- On fait tourner chacun des quartiers d'un quart de tour, de façon récursive.



Cela donne les étapes suivantes avec l'image `chien-rotation.png` :



Exercice 3 (sur papier) : Pour faire les déplacements, il faut permuter les pixels d'un quartier avec ceux à la même position dans les autres quartiers.
Déterminer les coordonnées des autres pixels correspondant à celui de coordonnées `(x, y)`.



Exercice 4 (sur papier) : Déterminer les coordonnées des coins haut gauche des 4 quartiers.

Exercice 5 : Compléter le code de la fonction `rotation_dpr2(pixels, xmin, ymin, N)` qui fait la rotation du quartier dont le coin haut gauche est en `(xmin, ymin)` et dont les côtés mesurent `N` pixels.

```
def rotation_dpr2(pixels, xmin, ymin, N):
    if N > 1:
        # On coupe en deux
        N = N // 2
        # On fait les déplacements
        for x in range(xmin, xmin+N):
            for y in range(ymin, ymin+N):
                tmp = pixels[x, y]
                pixels[x, y] = pixels[..., ...]
                pixels[..., ...] = pixels[..., ...]
                pixels[..., ...] = pixels[..., ...]
                pixels[..., ...] = tmp
        # On fait les rotations des 4 quartiers
        rotation_dpr2(pixels, ..., ..., N)
        rotation_dpr2(pixels, ..., ..., N)
        rotation_dpr2(pixels, ..., ..., N)
        rotation_dpr2(pixels, ..., ..., N)

def rotation_dpr(nom):
    image = Image.open(nom)
    N = image.width  # largeur de l'image
    pixels = image.load()
    rotation_dpr2(pixels, 0, 0, N)
    image.show()
```

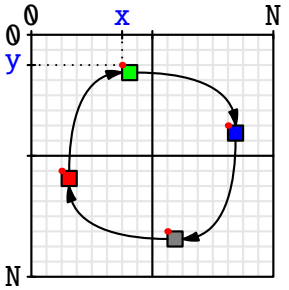
```
rotation_dpr("chien-rotation.png")
```

Pour voir les étapes, vous pouvez changer la valeur du test `if N > 1` par n'importe quelle puissance de 2.

Rotation efficace

La méthode divisier pour régner donne une façon originale de faire la rotation, mais chaque pixel est déplacé $\log_2(N)$ fois, où N est le nombre de pixels sur le côté. Ce n'est pas si efficace que cela. Il est possible de ne déplacer chaque pixels qu'une seule fois en faisant les mettant directement à la bonne place. La méthode que nous allons utiliser ne fonctionne que si l'image est carrée et que le nombre de pixels sur le côté est pair.

Exercice 6 (sur papier) : Déterminer les coordonnées des autres pixels faisant partie de la permutation avec celui de coordonnées `(x, y)`.



Exercice 7 : Compléter la fonction `rotation_efficace(nom)`.

```
def rotation_efficace(nom):
    image = Image.open(nom)
    N = image.width
    pixels = image.load()
    for x in range(N//2):
        for y in range(N//2):
            tmp = pixels[x, y]
            pixels[x, y] = pixels[..., ...]
            pixels[..., ...] = pixels[..., ...]
            pixels[..., ...] = pixels[..., ...]
            pixels[..., ...] = tmp
    image.show()
```