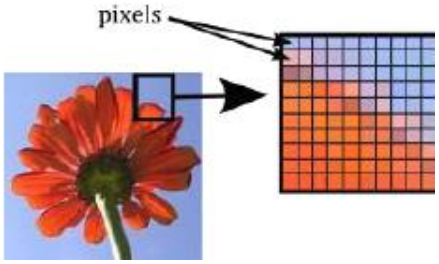
I Représentation d'une image

Une image numérique est composée de pixels, arrangés en un rectangle.



Le pixel en haut à gauche est numéroté (0,0), celui en bas à droite $(H-1, L-1)$ si l'image est de hauteur H et de largeur L. La convention d'ordre des indices est la même que pour les matrices.
Dans notre cas, chaque pixel est décrit par la donnée de trois intensités de couleurs R, G et B qui sont des entiers entre 0 et 255. Ainsi chaque pixel est codé par une liste de longueur 3 [R, G, B] composée d'octets.
Nos objets images en python seront donc des listes de lignes de pixels c'est à dire des listes de listes de pixels ou encore des listes de listes de listes...
Pour charger une image depuis un fichier, nous disposons de la fonction `lit_image(chemin)` qui prend une chaîne de caractère représentant le chemin du fichier à charger comme argument.

```
image = lit_image('le chemin vers le fichier')
image[0][0] # retourne le premier pixel, une liste de longueur 3
image[0][0][1] # retourne la composante verte
```

Les images en noir et blanc (niveaux de gris) sont représentés par des pixels ayant des intensités égales pour chaque couleur.

II Manipulations simple

Taille de l'image

On considère qu'une image a été créée avec le code précédent.

1. Donner une instruction python (sur feuille) permettant de connaître la hauteur de `image` (le nombre de lignes)
2. Idem pour la largeur qui est le nombre de colonnes.

Negatif

Charger chacune des images proposées avec ce TD et utiliser la fonction `affiche_image` sur l'image calculée comme `255-image` (si votre image s'appelle `image`).
Sachant que `lit_image` retourne un tableau numpy de pixels, quelle est l'opération effectuée lorsque l'on calcule

```
255 - image
```

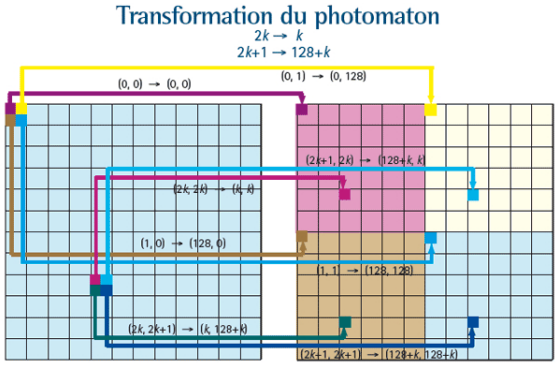
Rotation

Une des images proposée a été retournée. on se propose ici d'étudier l'opération à effectuer pour retrouver l'image d'origine.

1. On considère une image de taille H × L. Posons $(i, j) \in [\![0, H-1]\!] \times [\![0, L-1]\!]$. On considère le pixel de position (i, j) dans l'image de départ. Quelle est sa position dans l'image retournée ?
2. Si (i, j) est la position du pixel dans l'image retournée, quelle est sa position dans l'image de départ ?
3. Compléter la fonction `rotation` et tester le résultat.

III Photomaton

La transformation considérée ici est illustrée sur une image de taille 256 × 256 :



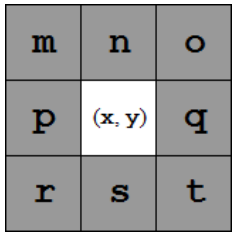
1. Notre image (lena.png) est de taille 512 × 512. Que deviennent les formules de transformation ?
2. Rappeler comment on détecte si un entier est pair ou impair en python.
3. Implémenter cette transformation dans la fonction `photomaton` et l'effectuer 9 fois sur notre image.

IV Détection de contour

Pixels "proches"

On considère une image de taille H × L et un pixel en position (x, yj) qui n'est pas au bord de l'image (donc $x, y \neq 0$ et $x \neq \mathrm{H} - 1$ et $y \neq \mathrm{L} - 1$)

1. Les pixels voisins que nous considérerons sont les pixels notés n, p, q, s sur la figure.



2. Chaque pixel est donné par 3 couleurs (r, v, b) que l'on indicera par son nom. Par exemple le pixel noté n est (r_n, v_n, b_n).
3. La distance entre deux pixels est définie comme la norme de la différence de ces pixels considérés comme des vecteurs. Ainsi la distance de n à s est

$$d(n, s) = \sqrt{(r_n - r_s)^2 + (v_n - v_s)^2 + (b_n - b_s)^2}$$

4. On définit la distance du pixel de position (x, y) à ses voisins par

$$l = \sqrt{d(n, s)^2 + d(p, q)^2}$$

Calcul des distances

1. Compléter la fonction `distance` qui prend comme argument une image et une position de pixel et retourne la distance l telle que définie au paragraphe précédent. On admet dans cette fonction que le pixel considéré n'est pas au bord.

   Astuce : on pourra utiliser le fait que les pixels sont en fait des vecteurs numpy et donc que les opérations $+, *, **, \ldots$ se font terme à terme. De plus, pour sommer les éléments d'une liste ou d'un vecteur, on peut utiliser la fonction `sum`.

2. Compléter la fonction `tableau_distances` qui prend comme argument une image `img` de taille H × L et retourne un tableau numpy de taille $(\mathrm{H} - 2) \times (\mathrm{L} - 2)$ qui contient la distance de chaque pixel qui ne soit pas au bord de `img` à ses voisins.

   La distance d'indices $(0, 0)$ dans le tableau des distances sera la distance du pixel de position $(1, 1)$ à ses voisins dans l'image `img`.

Normalisation

On souhaite maintenant convertir le tableau des distances que l'on a calculé en image en noir et blanc.

- Étape 1 : on calcule (en utilisant `np.max`) le maximum des distances obtenues, notée l_{max}
- Étape 2 : on remplace chaque valeur l calculée par $l_{norm} = \lfloor 255 \frac{l}{l_{max}} \rfloor$ pour obtenir un entier entre 0 et 255.
  pour calculer la partie entière, on utilise `int` qui converti un nombre flottant (ou une chaîne de caractère) en entier.
- On crée l'image en noir et blanc correspondante : chaque pixel est `[`l_{norm}`,` l_{norm}`,` l_{norm}`]` qui sera de taille $(\mathrm{H} - 2) \times (\mathrm{L} - 2)$.

Pour l'image **nuages.png** on obtient