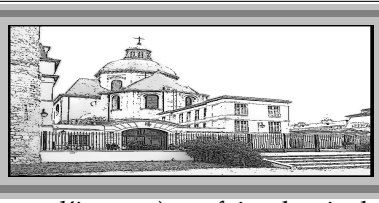
Table des matières

0.1 Image en noir et blanc. . . . 2
0.2 Images en niveaux de gris. Création d'images . . . 3
0.3 Images en niveau de gris, inversion d'images . . . 4
0.3.1 Variante : . . . 4
0.4 Images en couleur. Lecture et traitement d'images. . . . 4
0.4.1 Habillez le père Noël en vert. . . . 6
0.4.2 Habillez Jésus. . . . 6
0.4.3 Collez un Sucri devant Jesus . . . 6
0.5 De la couleur au noir et blanc . . . 6
0.5.1 Retouche d'image noir et blanc. . . . 7
0.5.2 Floutage d'image noir et blanc . . . 7
0.6 Sténographie . . . 7
0.7 Transformation du photomaton . . . 8
0.7.1 Remarque fondamentale : . . . 8
0.7.2 Variante : . . . 8

LYCEE CHARLEMAGNE

Mardi avril

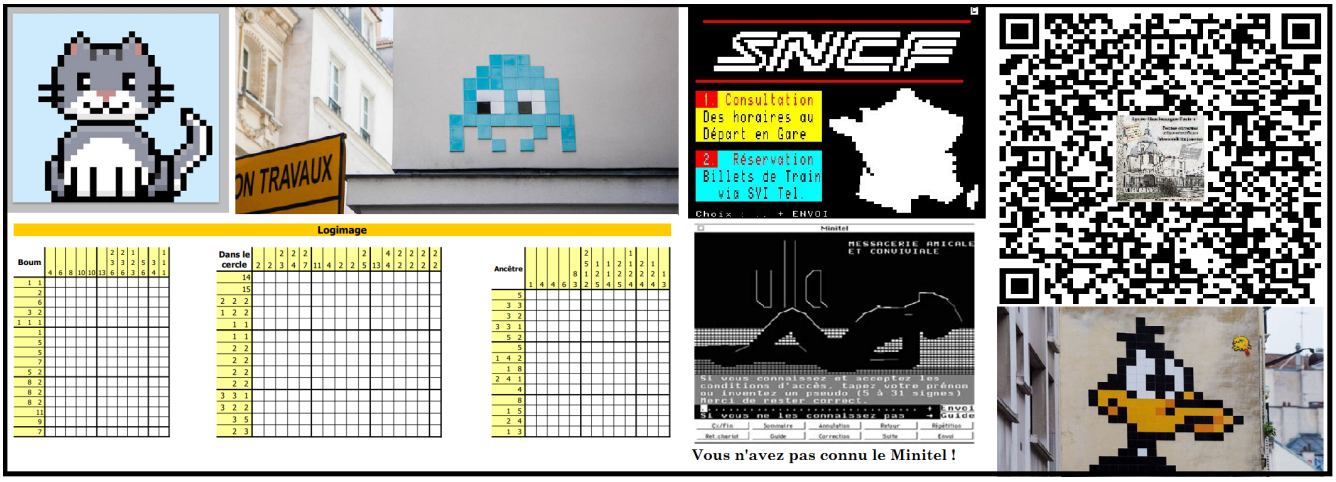
*M.P.S.I.*2



2024 2025

Info

Une image (en informatique et en traitement d'images) est faite de pixels (PICTURE ELEMENT), disposés en liges, les uns à côté des autres.
D'où l'usage de matrices (ou de tableaux).



Une fois n'est pas coutume, on va utiliser `numpy`, car le transfert d'un tableau traité numériquement à un tableau de pixels passe bien avec des fonctions.

Ce qui change :
• créer un tableau à deux dimensions à `n` lignes et `q` colonnes dont les éléments sont des entiers à `8 bits` (donc de `0` à `256`)

```
gris = np.zeros((n, q), dtype = np.uint8)
```

• créer un tableau à deux dimensions à `n` lignes et `q` colonnes dont les éléments sont des triplets d'entiers à `8 bits` (R, G et B vous connaissez ?)
```
couleur = np.zeros((n, q, 3), dtype = np.uint8)
```

• extraire l'élément de ligne `i` et colonne `q` d'un tableau `T` : `T[i, k]` et pas `T[i][k]`

• obtenir la taille d'un tableau `T.shape` (si le tableau est d'une seule dimension, la réponse est un entier, si c'est un tableau matriciel, la réponse est un couple, si c'est un tableau à trois dimensions, la réponse est un triple)

Et on va utiliser une partie du module `PIL`[1].

Importer le module	`from PIL import Image`
Lire un fichier `jpg` sur le disque et le transférer dans un objet `PIL`	`papanoel = Image.open("perenoel.jpg")`
Transformer ce fichier `PIL` en tableau `numpy`	`tableau = np.array(papanoel)`
Transformer un tableau `numpy` (ou pas) en fichier `PIL`	`im = Image.fromarray(tableau)`
Afficher un fichier `PIL`	`im.show()`
Sauvegarder un fichier	`im.save('SaintNicolas.jpg')`

Cette image sera affichée sur une fenêtre graphique. Pensez à fermer vos fenêtres de temps en temps.

Dès le début de votre programme, vous tapez donc pour être tranquilles :

```
import numpy as np
from PIL import Image
from math import *
from random import *
```

Vous aurez peut être besoin d'utiliser des fonctions mathématiques.

Ou alors vous allez créer des motifs aléatoires.

0.1 Image en noir et blanc.

Un peu de hors sujet théorique que vous pouvez sauter, mais qui va passionner Raphaël et d'autres.

Un logimage consiste en une grille de `nl` lignes et `nc` colonnes, dans laquelle il faut retrouver une image en noir et blanc à partir de clés. Chaque ligne (et chaque colonne) est codée par une liste de clés entières. Un entier m dans une liste de clés correspond à un bloc de m cases consécutives à colorier en noir. Par exemple, pour une ligne donnée, la liste de clés [2, 4] signifie qu'en regardant les cases de cette ligne de gauche à droite, on trouve d'abord un certain nombre (éventuellement nul) de cases blanches, puis un bloc de 2 cases noires consécutives, suivi d'au moins une case blanche, puis un bloc de 4 cases noires, puis éventuellement des cases blanches. De même, la liste de clés pour une colonne décrit la taille des blocs rencontrés depuis le haut jusqu'au bas de la colonne. On veut s'assurer que la solution d'un logimage existe et est unique. La figure 1 présente un exemple de logimage et sa solution : les listes en regard des lignes et des colonnes sont les clés codant la solution. Par exemple, la deuxième ligne est codée par une liste de deux clés : [3, 1], alors que la première ligne est codée par une liste contenant une seule clé : [2]. [1] [5] [4] [2] [4]

Si vous voulez, le sujet X-ENS de MP / MPI / PC / PSI Informatique B traite de la résolution des logimages.
https://www.polytechnique.edu/admission-cycle-ingenieur/sites/admission/files/content/2024%20MP-MPI-PC-PSI%20SUJET%20INFORMATIQUE%20B%20(XELSR).pdf

1. non, pas PUBLIC IMAGE LIMITED, le groupe fondé par JOHNNY ROTTEN après les SEX-PISTOLS ; c'est juste `Python Imaging Library`

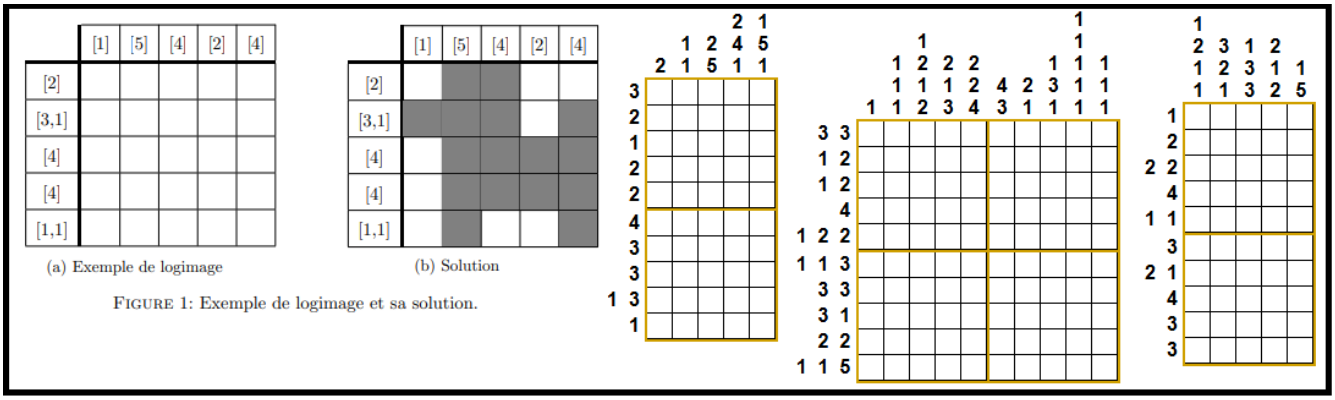


FIGURE 1: Exemple de logimage et sa solution.

Le logimage « petit chien » du sujet correspondrait au tableau suivant
`t = [[255, 0, 0, 255, 255], [0, 0, 0, 255, 0], [255, 0, 0, 0, 0],...]`

et serait visualisé par l'instruction suivante : `im = Image.fromarray(t)` puis `im.show()`.

Mais l'image serait très petite à l'écran, quelques pixels.

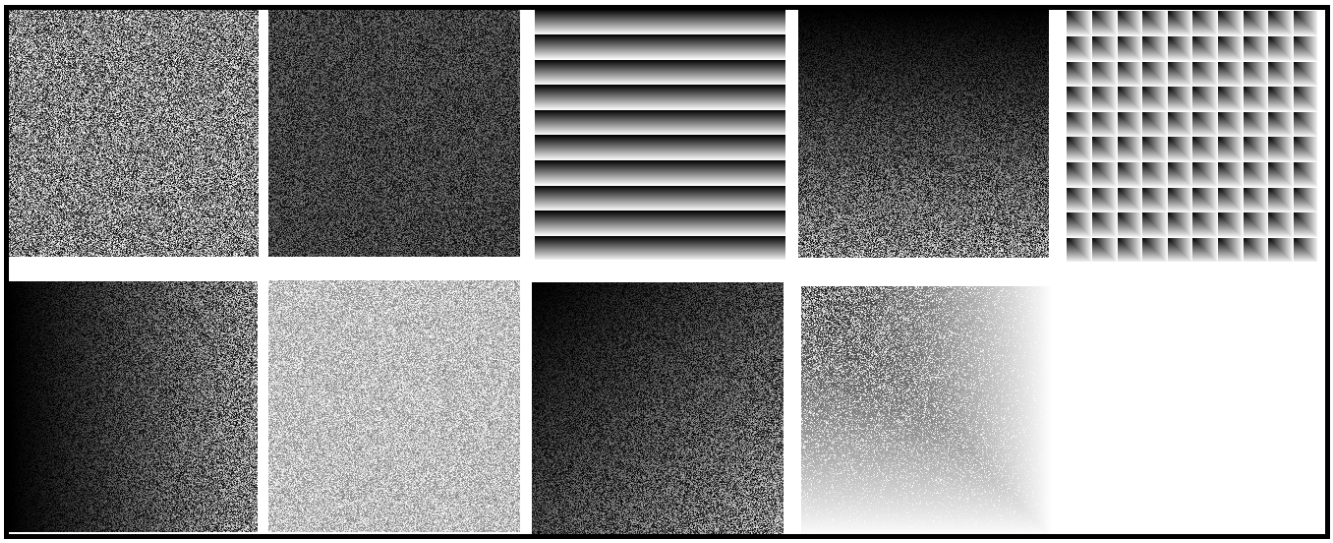
0.2 Images en niveaux de gris. Création d'images

Créez un `np.array` de taille `256` sur `256` d'abord empli de `0`.
Remplissez le avec les éléments sont des entiers tirés au hasard entre `0` et `256` (double boucle `for`, affectation `T[i, k]` à `randrange(256)`).
Affichez le.

Recommencez mais avec des instructions plus intelligentes.

Retrouvez dans les fonctions suivantes insérées dans la double boucle for celles qui ont donné les images ci-dessous.
S'il y a plus d'images que fonctions, plus de fonctions que d'images ou des associations non faites, complétez.

A	`randrange(256)`	B	`randrange(128)`	C	`(10*k)%256)`	D	`randrange(128,256)`
E	`randrange(i+1)`	F	`randrange(k+1)`	G	`(10*i) % 256`	H	`max(randrange(256), i)`
I	`max(((10*i) % 256)`	J	`255`	K	`randrange((i+k)//2+1)`	L	`max(randrange(256),i,k)`



Que vont donner

`max(i, k)`	`max(i, 255-k)`	`min(i, k)`	`(i+k) % 256`	`int(i>k) * 128`	`(i*k) % 256`

Je veux un disque de centre `128, 128`, de rayon `80` en blanc sur un fond noir. Vous utiliserez la comparaison de

`sqrt((i-128)**2+(k-128)**2)` avec `80`, sauf si savez vous passer de `sqrt`, parce que vous êtes matheux.

Je veux un dégradé circulaire où le centre `128, 128` est blanc et les points du bord `(0, 0), (0, 255), (255, 255)` et `(255, 0)` sont noirs. Vous utiliserez encore `sqrt((i-128)**2+(k-128)**2)` en vous souvenant qu'il faut que le résultat soit un entier (et qu'il ne doit pas atteindre `256`).

0.3 Images en niveau de gris, inversion d'images

• Ouvrez une image déjà existante, comme la cour du lycée `cour = Image.open(...)`.

• Tranférez dans un tableau `numpy`.

• Parcourez ce tableau et inversez le gris de chaque pixel (d'abord simple x -> 256-x).

• Retransformez en image `im = Image.fromarray(...)`.

• Affichez la avec `im.show( )`.

0.3.1 Variante :

Appliquez une fonction mathématique qui va accentuer les effets, comme `x -> (x*x) // 256`
`x -> int((1.5 + atan((x-128) / 16)) * 80)`
ou toute autre fonction à inventer.

0.4 Images en couleur. Lecture et traitement d'images.

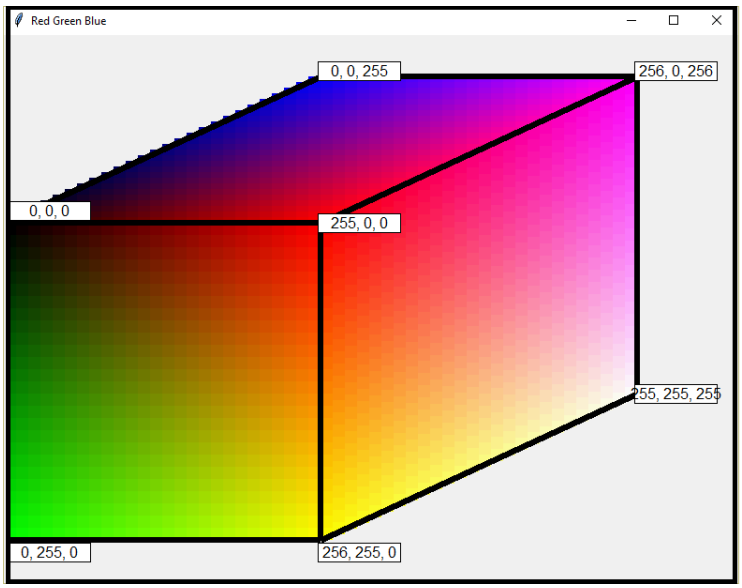
Regardez de près une affiche dans le métro. De loin, vous voyez une image en couleurs. De près, vous voyez un traitement point par point, et chaque point est fait lui même de petites pastilles des trois couleurs fondamentales, plus ou moins grandes. La taille de ces pastilles fait que chacune des trois couleurs sera plus ou moins présente. La présence plus ou moins fortes de Rouge, Vert et Bleu fera que vous percevrez une couleur intermédiaire, sur le cercle chromatique.
La densité des couleurs des pastilles donnera enfin la densité de la couleur de synthèse obtenue.

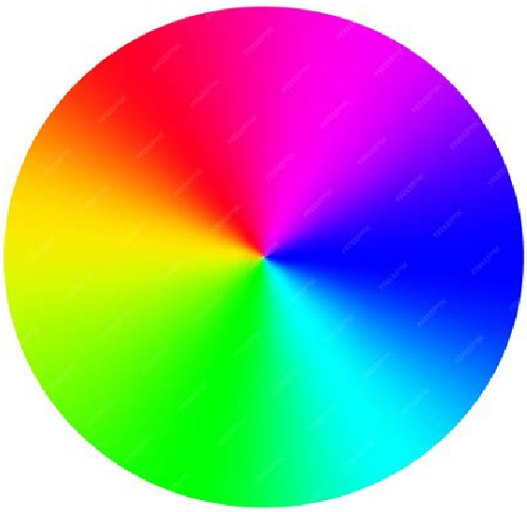
On va donc faire de même informatiquement, avec des pixels codés par un triplet d'entiers et non plus un seul entier de 0 à 256.

<table>
<tr><td rowspan="4">[0, 0, 0]

noir</td><td>[256, 0, 0]</td><td>[124, 0, 0]</td><td></td><td>[0, 256, 0]</td><td>[0, 128, 0]</td><td></td><td>[0, 0, 256]</td><td>[0, 0, 128]</td><td rowspan="4">[255, 255, 255]

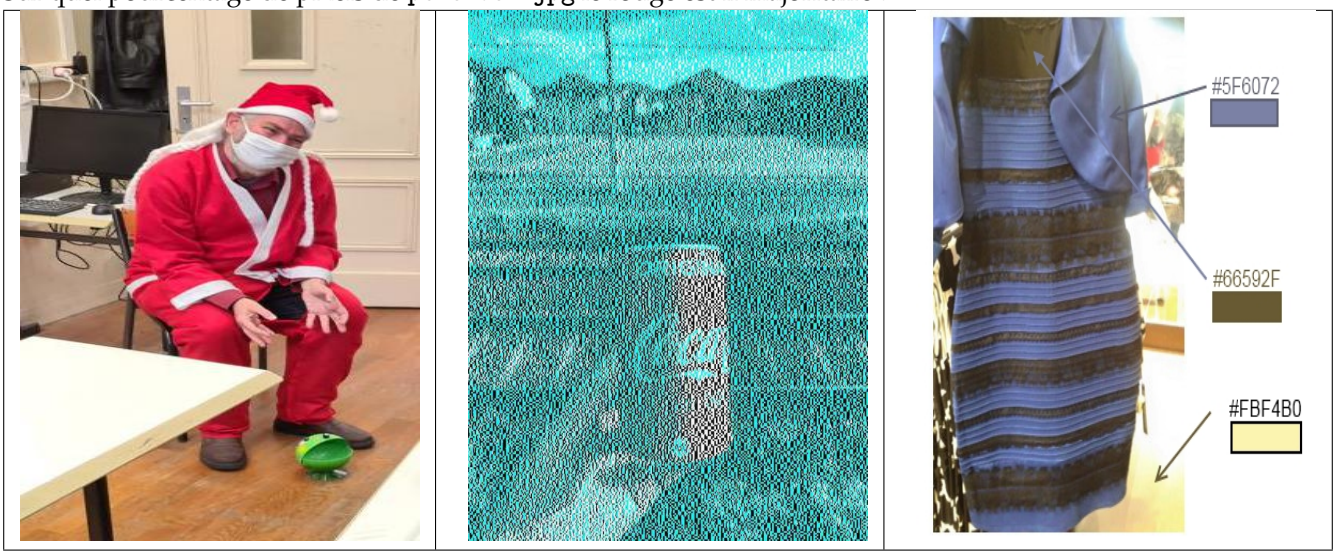
blanc</td></tr>
<tr><td>rouge
pétant</td><td>rouge
sombre</td><td>[255, 255, 0]</td><td>vert
claquant</td><td>vert
bouteille</td><td>[0, 255, 255]</td><td>bleu
profond</td><td>bleu
nuit</td></tr>
<tr><td></td><td></td><td>jaune</td><td></td><td></td><td>cyan</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td colspan="3">[255, 0, 255] : magenta</td><td></td><td></td></tr>
</table>





Evidemment, la version noir et blanc n'est pas géniale.

Sur quel pourcentage de pixels de `perenoel.jpg` le rouge est il majoritaire ?



PereNoel.jpg	CanetteCocaRouge.jpg	Robe.jpg

Sur quel pourcentage de pixels de `canettecoca.jpg` le rouge est il majoritaire ?

Et le pourcentage de bleu dans la robe qui a soulevé la polémique sur les réseaux sociaux il y a quelques années ?

• Ouvrez le fichier, transférez le dans un tableau.

• Pour récupérer le nombre de lignes et de colonnes d'un tableau `numpy` à deux dimensions `tab` : `nl, nc = np.shape(tab)`, la fonction `shape` récupère un couple

• Parcourez les cases du tableau
```
for i in range(nl) :
....for k in range(nc)
........p = tab[i, k]
```
et pour chaque pixel p, regardez si `p[0]` (rouge) est plus grand que 128
- la somme des deux autres
- le maximum des deux autres
- le double du maximum des deux autres

choisissez le critère qui vous semble le plus pertinent.

• Divisez ensuite par le nombre total de pixel.

0.4.1 Habillez le père Noël en vert.

• Chargez l'image `perenoel.jpg`, faites en un tableau numpy `christmas` (par exemple).
• Parcourez par double boucle les pixels du tableau `p = christmas[i, k]`.
• Détectez les pixels rouges (par un critère comme à la question précédente), et faites en des pixels verts (en échangeant par exemple deux des composantes de `p`).
• Re-transformez `christmas` en image, et affichez le. [2] .

> Attention, si votre critère de détection de la couleur rouge n'est pas assez pertinent, vous allez me transformer en HULK [3]. Essayez.

Refaites le avec une autre couleur de remplacement.
Allez chercher le teletubies `teletub.png` et colriez le dans la couleur de votre choix.

> Remarque : ce teletubies est en format png, moins compressé que jpg, avec quatre coefficients matriciels : R, G, B et facteur alpha (de transparence).

0.4.2 Habillez Jésus.

Chargez cette fois l'image `JesuisSolal.jpg`. Détectez les pixels de sa toge et changez leur couleur.

Ou alors détectez les pixels très blancs (mais ils sont trop nombreux dans l'image) et changez les de couleur.

0.4.3 Collez un Sucri devant Jesus

Ouvrez deux images avec donc deux noms différentes :
`jesus = Image.open('Jesuisolal.jpg')` et `dieu = Image.open('PereNoel.jpg')`.

Remplacez sur une zone
```
for i in range(debuti, fini) :
....for k in range(debutk, fink) :
```
les pixels de `jesus` par ceux de `dieu` d'une zone de même taille.
Affichez la nouvelle image `Image.fromaaary(jesus).show()`.

Si vous êtes expert, ne le faites que si le pixel issu de `dieu` est d'une couleur verte majoritaire.

0.5 De la couleur au noir et blanc

Ouvrez l'image couleur de `JésusuisSolal` et faites en un tableau `TCoul` (les pixels sont des listes de trois entiers : `RGB`).

Créez un tableau `numpy` de même taille `TNB`, mais dont les pixels sont un seul entier (niveau de gris) (utilisez `shape` et `np.zeros`).
Parcourez par double boucle les pixels de `TCoul`, et remplissez les pixels de `TNB` avec l'une des règles suivantes par exemple, pour voir l'effet dans chaque cas :

`r = TCoul[i, k, 0]`
`g = TCoul[i, k, 1]`
`b = TCoul[i, k, 2]`
	`pnb = (r+g+b) // 3`	`pnb = max(r, g, b)`	`pnb = min(r, g, b)`

`pnb = TNB[i, k]`	`pnb = if (r+g+b) > 360 :`
	`....255`
	`elif ...`
			`abs(r - g)`

Trouvez une solution pour accentuer les contrastes avec des `if`.

Sauvegardez l'image que vous trouvez le plus jolie, elle peut re-servir. Sous un nom pas encore utilisé.

2. si vous voulez le faire en one-liner avec `Image.fromarray(christmas).show()`, c'est comme vous voulez, mais moi je perçois ça comme une invocation magique dans laquelle je me perds comme avec `mathplotlib` qui n'a de maths que le nom.
3. ok, HULK, c'est vieux, seuls vos parents ont la ref, pareil pour le GÉANT VERT, peut être SCHREK, mais là, ça me vexe un peu

0.5.1 Retouche d'image noir et blanc.

Vous avez sauvegardé une image Solal.jpg en niveaux de gris ?
On en envie de détecter les contours pour les accentuer.
Comment interpréter un contour ? C'est un passage « brutal » d'une gris très clair à un gris plus foncé par exemple.
Vous allez donc transférer votre image dans un `np.array` appelée disons `old` et créer un `np.array` appelé disons `new`, de même taille mais avec des `0`.
Vous allez recopier par double boucle [4] les pixels de `old`, sauf ceux pour lesquels un certain critère est vérifié. Et pour ceux là, vous mettrez `new[i, k]` à la valeur `255` par exemple.

Quel critère appliquer ?
- `old[i, k]` a une différence d'au moins `40` (par exemple) avec l'un des quatre voisins `old[i-1, k]`, `old[i+1, k]`, `old[i, k-1]`, `old[i, k+1]`
- `old[i, k]` a une différence d'au moins `40` (par exemple) avec au moins deux des quatre voisins `old[i-1, k]`, `old[i+1, k]`, `old[i, k-1]`, `old[i, k+1]`
- `old[i, k]` a une différence d'au moins `40` (par exemple) avec l'un des quatre voisins en ligne `old[i-1, k]`, `old[i+1, k]`, `old[i-2, k]`, `old[i+2, k]`

autre critère.

0.5.2 Floutage d'image noir et blanc

Et si on remplaçait chaque pixel de `Solal.jpg` par la moyenne de ses quatre presque voisins ?
On va essayer.

Mais pourquoi ne pouvez vous pas faire ce qui suit (il y a plusieurs erreurs) :

```
for i in range(nl) :
....for k in range(nc) :
........t[i, k] = (T[i-2, k] + T[i+2, k] + T[i, k-2] + T[i, k+2]) / 4
```

Comment rectifier ces erreurs ?

Sachant que numpy est très efficace pour faire des produits matriciels, avec vous une méthode pour effectuer ce calcul avec la matrice t et une matrice lissage de même format avec plein de 0 et de 1 ?

Et si vous appliquez plusieurs fois ce lissage ?

0.6 Sténographie

Le but du jeu : cacher une information dans une image, en superposant l'information à l'encodage de l'image, en dégradant l'image tellement peu que ceci ne se verra pas.

Si on remplace un pixel sur soixante quatre de l'image initiale, et en le remplaçant par un pixel de la nouvelle image (certes bien plus petite), on a « un petit défaut, mais on ne voit pas ce qui est caché.

Dans l'image `tele2.png` (taille sur `512` sur `256`), les pixels `T[i, k]` avec `i` congru à `a` modulo `8` et `k` congru à `b` modulo `4` ne sont pas les vrais.

Leur composante `T[i, k, 0]` a été remplacée par le niveau de gris d'une image mystère.

Pour les retrouver, recopiez en position `[i, k]` d'un nouveau tableau (de quelle taille ?) l'information du pixels de position `[8*i+a, 8*k+b]` de la version `np.array` d'ESTEBAN.

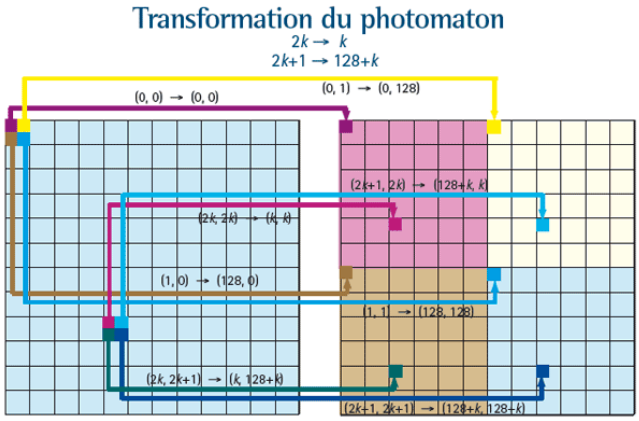
Transformez en image et affichez.

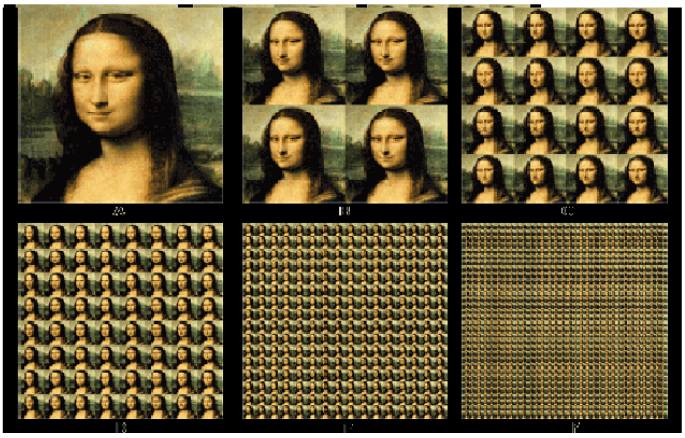
4. attention à votre `range`, vous allez comprendre

Ah pardon ? Vous ne connaissez pas `a` et `b` ? C'est normal.
A vous de les trouver.

Et si maintenant vous cachiez vous même une image derrière Esteban et l'envoyiez à un de vos camarades avec des éléments de la clef pour la retrouver ?

0.7 Transformation du photomaton





On prend une image initiale et en crée quatre copies plus petites qu'on dispose comme les quatre photos d'une planche de photomaton.

Mais aucun pixel de l'image initiale n'est perdu, donc aucune information.

En effet, on découpe l'image initiale en carrés de quatre pixels, et c'st chacun de ces quatre pixel qui va vers chacune des quatre photos.
De fait, vous pouvez raconter les choses avec des formules explicites de destination de chaque pixel initial

`p[2*i, 2*k]` va en `[i, k]`	`p[2*i, 2*k+1]` va en `[..., ...]`
`p[2*i+1, 2*k]` va en `[..., ...]`	`p[2*i+1, 2*k+1]` va en `[..., ...]`

Il existe de très belles formules basées sur la transformation du nombre binaire abcde en eabcd par exemple, je vous laisse lire interstices à ce sujet.

0.7.1 Remarque fondamentale :

Comme aucun pixel n'est perdu, la transformation est une bijection de l'ensemble des pixels de l'image.
On dira même une permutation sur un ensemble à longueur x largeur éléments.
Et comme toute permutation, elle a un ordre.

Si vous appliquez la transformation un nombre suffisant de fois, vous revenez à l'image initiale.
Faites le, en affichant à chaque fois la nouvelle image (avec des images de format 2^p sur 2^q, la période n'est pas trop longue.

Faites le pour `perenoel.jpg`, et aussi sur `tele2.png`, car là, il y a une surprise finalement.

0.7.2 Variante :

Débrouillez vous pour qu'une des quatre images du photomaton ait « la tête en bas ».

Débrouillez vous pour qu'une des quatre images du photomaton ait « la tête en bas ».