Partie 4 : La simulation numérique

Chapitre 3

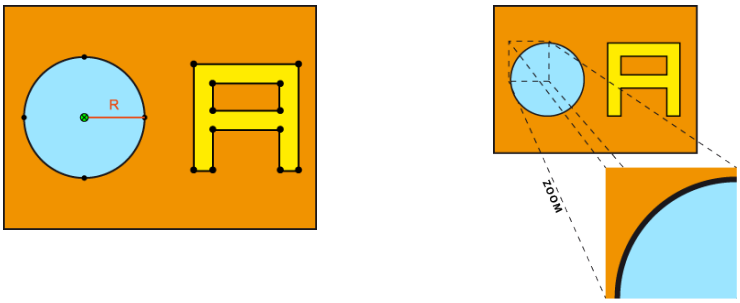
Traitement de l'image

Introduction :

On désigne sous le terme d'image numérique toute image (dessin, icône, photographie …) acquise (scanners, appareils photo, …), créée (par des programmes informatiques, via la souris, …), traitée ou stockée sur un support informatique (disque dur, CD-ROM, …) sous forme binaire (suite de 0 et de 1). Il existe deux sortes d'images numériques : Les images Vectorielles et les images Matricielles.

1. Les images Vectorielles

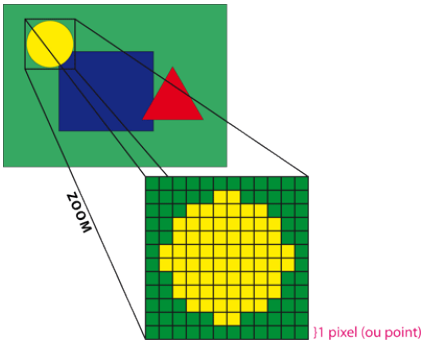
Une image vectorielle en informatique, est une image numérique composée d'objets géométriques individuels (segments de droite, polygones, arcs de cercle, etc.), définis chacun par différents attributs (forme, position, couleur, remplissage, visibilité, etc.). Par exemple, une image vectorielle d'un cercle seule la position du centre, la taille du rayon et ses informations de couleurs seront mémorisées.
Les images vectorielles présentent deux avantages : elles occupent peu de place en mémoire et peuvent être redimensionnées sans perte d'informations et sans effet dit : d'escalier.
Exemple :



On observe que lorsque l'on zoom sur l'image, la ligne du bord du cercle reste lisse, il n'y a pas d'effet escalier.
Il existe de nombreux formats de fichiers vectoriels : PDF, Illustrator, Flash, SVG… et de nombreux logiciels de dessin vectoriel : Illustrator, InDesign, Autocad, Flash,…

2. Les images matricielles (Bitmap)

Une image matricielle est formée d'un assemblage de points nommés pixels. Un pixel (picture element) est le plus petit élément d'une image.
Exemple :



Dans le cadre de notre cours de traitement d'images en Python, nous étudierons seulement les images matricielles.

Propriétés des images matricielles

1. La taille exprimée en pixels qu'on appelle définition

On appelle « **définition** » le nombre de pixels constituant l'image, c'est-à-dire sa «dimension informatique» : le nombre de colonnes de l'image que multiplie son nombre de lignes.

Une image de taille 1920 pixels de largeur (nombre de colonnes) et 1080 pixels d'hauteur (nombre de lignes) aura une définition :
1920 ×1080 = 2 073 600 pixels = 2.07 Mégapixels = 2.07 M px

2. Le codage de la couleur

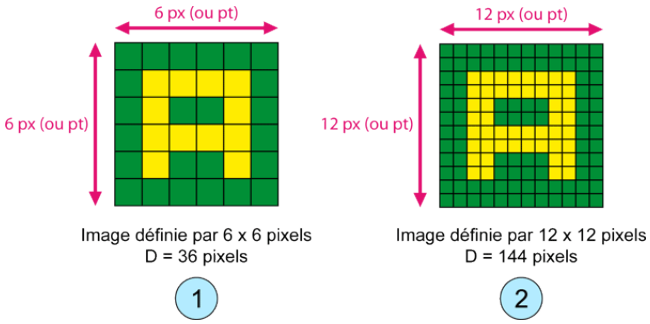
Une image est donc représentée par un tableau à deux dimensions dont chaque case est un pixel. Pour représenter informatiquement une image, il suffit donc de créer un tableau de pixels dont chaque case contient une valeur. La valeur stockée dans une case est codée sur un certain nombre de bits déterminant la couleur ou l'intensité du pixel. Les images informatisées se présentent en plusieurs niveaux de couleurs. Une image peut être en couleurs, en niveaux de gris, en noir et blanc.

Mode d'image	Nombre de couleurs possibles	Mode d'image en Python
1 bit	2 (valeur 0 : Noir et valeur 1 : Blanc).	**1**
8 bits ou mode **Niveaux de gris**	256 couleurs où la valeur (luminosité) de chaque pixel est comprise entre 0 et 255 (avec 0 : Noir et 255 : Blanc) et codée sur un octet (8 bits). Exemple : 00000000 codes le Noir, 11111111 codes le Blanc.	**L**
24 bits ou mode **RVB** ou vraies couleurs	16 777 216 couleurs. Chaque pixel est un mélange de trois teintes : rouge, vert et bleu. La valeur de chaque teinte est comprise entre 0 et 255 et codée sur un octet. Exemple : (109, 148, 114) = (01101101, 10010100, 01110010)	**RGB**

3. Le poids

Le poids de l'image dépend de la quantité de pixels constituant l'image, c'est le pixel qui en informatique a un poids.

Plus la densité des pixels constituant l'image matricielle est élevée, plus le nombre d'informations est grand, plus l'image est définie, mais aussi plus le poids de l'image est grand.



Le poids (taille) de l'image est alors égal à son nombre de pixels que multiplie le poids de chacun de ces éléments.

Taille du fichier image = Largeur × Hauteur × Nombre de bits de couleur par pixel

Voici le calcul pour une image en couleur d'une définition de 640 × 480 pixels :

Nombre de pixels: $640 \times 480 = 307\,200\ pixels$
Poids de chaque pixel: (1 octet = 8 bits)
Le poids de l'image est ainsi égal à : $307\,200\ pixels \times 3\ octets = \frac{921600\ octets}{1024} = 900\ ko$

Pour connaître la taille en Ko il suffit de diviser par 1024.

Définition de l'image	Noir et blanc (1 bit / pixel)	256 couleurs ou nuances de gris (8 bits / pixel)	Couleurs réelles : 16 000 000 couleurs (24 bits / pixel)
320x200	7.8 Ko	62.5 Ko	187.5 Ko
640x480	37.5 Ko	300 Ko	900 Ko
800x600	58.6 Ko	468.7 Ko	1.4 Mo
1024x768	96 Ko	768 Ko	2.3 Mo

Plus le nombre de pixels de l'image matricielle est élevé, plus la place occupée en mémoire (sur le disque dur, …) sera élevée, mais aussi la durée de traitement sera importante.

A) Traitement basique des images matricielles en Python

Il existe plusieurs modules permettant de manipuler une image sous Python : le module PIL ,le module ndimage de Scipy, matplotlib.image, ImageWindow, etc. Durant notre cours nous étudierons les méthodes de traitement d'image définies dans la classe Image du module PIL.

Le module Python « PIL » de traitement d'image

Le module PIL (Python Image Library) permet d'ouvrir/créer des fichiers images et de les manipuler (accès à la valeur d'un pixel, modification d'un pixel, etc.). C'est le module phare de traitement de l'image en Python. D'autres informations peuvent être trouvées dans la documentation de PIL : http://effbot.org/imagingbook/

Dans un script Python, on peut importer le module PIL de différentes manières :

```
>>> from PIL.Image import *
>>> import PIL.Image
>>> import PIL.Image as im
```

1) Ouverture et affichage d'une image

La fonction open() : ouvre un fichier image et crée un objet de type **Image**.

```
1.  from PIL.Image import *

2.  # Ouverture d'un fichier image et création d'un objet Image
3.  tiger = open ('tiger.jpg ')
4.  # Affichage de l'image dans une fenêtre
5.  tiger.show ()
6.  # Retour d'une séquence de valeurs des pixels
7.  data = tiger.getdata ()
8.  # Convertir la séquence des pixels en liste
9.  L = list (data)
10. print (L) # [(90 , 84, 96) , (80 , 73, 81) , ... , (240 , 231 , 172)]
11. print (len(L)) # 94208

12. # Récupérer la largeur et la hauteur ( taille ) de l'image
13. Largeur , hauteur = tiger.size ;
14. print ( Largeur , hauteur ) # 368 256 ; 368 * 256 = 94208 pixels
```

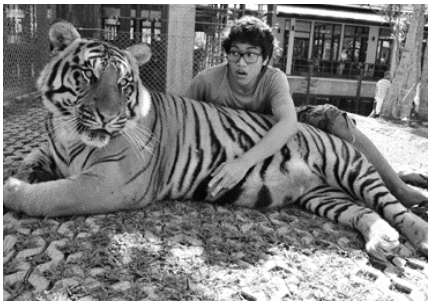
Exercice 1.
Écrire un script Python permettant d'ouvrir une image et compter le nombre de pixels parfaitement noirs et le nombre de pixels parfaitement blancs dans celle-ci.

2) Modifier le mode d'une image

```
1. # Retour une copie convertie d'image dans un mode différent
2. img_gris = tiger.convert ('L') # L : Mode 8 bits , valeurs de 0 à 255

3. # Affichage de l'image dans une fenetre
4. img_gris.show ()

5. # Sauvegarde de l'image dans un fichier
6. img_gris.save ('tigergris.bmp ')
```



3) Création d'une image

Par défaut, tous les pixels sont mis à 0. l'image est noire. Il est ensuite possible de modifier les valeurs des pixels pour créer une image en couleurs.

```
1. # Création d'un nouvel objet Image en précisant le mode et les
   dimensions
2. nouveau =new ('RGB ', ( Largeur , hauteur ))

3. # Affichage de l'image dans une fenêtre
4. nouveau.show ()
```

4) Remplissage d'une image

```
1. # Ouverture d'un fichier image et création d'un objet Image
2. tiger = open ('tiger.jpg ')

3. # Retour d'une séquence de valeurs des pixels
4. data = tiger.getdata()

5. # Remplissage de l'image « nouveau » à partir d'une liste de pixels
6. nouveau.putdata (list(data))
7. nouveau.show()
```

Exercice 2.

Écrire une fonction Python qui prend comme arguments deux entiers **n** et **p** et crée et affiche l'image d'un damier de dimensions n × p. Les pixels sont alternativement noirs et blancs le long des lignes comme le long des colonnes.

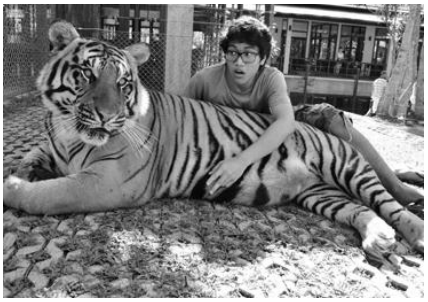
5) Modifier une image

Avant d'effectuer un traitement sur une image, il faut la décomposer en pixels.

On peut accéder aux pixels formant l'image de différentes manières :

- La méthode **load()** : retourne tous les pixels de l'image sous la forme d'un tableau à deux dimensions (liste de listes). Donc pour lire ou modifier la valeur d'un pixel il suffit de spécifier sa position (N° ligne, N° colonne).
- La méthode **getpixel((i ,j))** : retourne la valeur du pixel situé à la ligne i et à la colonne j.
- La méthode **putpixel((i ,j), valeur)** : modifie la valeur du pixel situé à la ligne i et à la colonne j.

Exemple :



```
1.  img_gris = tiger.convert('L')
2.  # Retour d'un tableau de pixels
3.  Tabpixel = img_gris.load()
4.  # Récupération de la valeur d'un pixel a une position donnée
5.  p1= img_gris.getpixel((0 ,1)) # == Tabpixel [0 ,1]
6.  print (p1) # 107
7.  # Modifier la valeur d'un pixel a une position donnée
8.  img_gris.putpixel((0 ,1) ,255)
9.  # en RGB : putpixel((0 ,1) ,(r,g,b))
10. # ou bien Tabpixel [0 ,1] = 255
11. p1= img_gris.getpixel ((0 ,1))
12. print (p1) # 255
13. img_gris.show ()
```

Exercice 3.
Écrire un script Python permettant d'inverser les niveaux de gris d'une image .

A retenir pour la suite des exercices :

- Le pixel en haut à gauche correspond au pixel de coordonnées (0,0).
- Le pixel en haut à droite est le pixel de coordonnées (largeur-1 , 0).
- Le pixel en bas à gauche a pour coordonnées (0 , hauteur-1).
- Le pixel en bas à droite correspond au pixel (largeur-1, hauteur-1).
- Pour une image en niveaux de gris, le code obtenu par *imageSource.getpixel((x,y))* est un nombre compris entre 0 et 255.
- Pour une image en couleurs, *p=imageSource.getpixel((x,y))* sera alors un tuple : p=(p[0],p[1],p[2]), ses trois composantes sont les composantes RVB, comprises entre 0 et 255.

Exercice 4.
Écrire un script Python permettant de **c**réer une image carrée (par exemple de taille 100 sur 100) ayant la forme d'une croix × noir sur fond blanc joignant les coins du carrée.

Avec le module *numpy*, on transforme directement l'image en tableau (ndarray).
- tabpixel = **numpy.array(**img**)** : retourne un tableau numpy de pixels à partir d'une Image PIL.
- img **= Image.fromarray(**tabpixel**)** : Créer une Image PIL à partir d'un tableau de pixels.

```
1.  import numpy
2.  import PIL.Image
3.  # Retour d'un tableau de pixels
4.  tabpixel = numpy.array ( img_gris )
5.  # Exemple d'un traitement simple
6.  for lines in tabpixel :
7.      for p in lines:
8.          p[0] ,p[1] = p[1] ,p[0]
9.  # Créer une image à partir d'un tableau de pixels
10. nouveau = Image.fromarray ( tabpixel )
11. nouveau.show ()
```

Exercice 5.
Écrire un script Python permettant d'ajouter autour d'une image une bordure d'épaisseur **e** pixels (e étant une variable du programme). Attention, cette bordure vient se placer autour de l'image, elle n'écrase pas les pixels existants.

6) Autres opérations sur les images

```
1. # Copie d'une image
2. tiger1 = tiger.copy ()
3. # Création d'une copie redimensionnée : resize((largeur, hauteur))
4. tiger2.resize ((120 ,120)).show ()
5. # Création d'une copie transposée
6. img_gris.transpose (FLIP_LEFT_RIGHT).show ()
7. # Création d'une copie retournée de l'image originale d'un angle en degrés à partir de son centre dans le
   sens contraire aux aiguilles d'une montre
8. img_gris.rotate (45).show ()
```

Exercice 6.
Transformer l'image de gauche en l'image de droite :

Exercice 7.
Écrire un script Python permettant d'ouvrir une image et d'enregistrer une copie agrandie d'un facteur d'agrandissement égal à 2, en hauteur comme en largeur. Chaque pixel de l'image originelle est donc remplacé par quatre pixels identiques.

Exercice 8.
Écrire un script Python permettant d'ouvrir une image et d'afficher l'image symétrique par rapport à un axe vertical. Même question en faisant subir cette fois à l'image une rotation de 90°.

B) Traitement avancé d'images matricielles en Python

1. La compression d'une image

La compression d'une image consiste à transformer une suite de bits A en une suite de bits B plus courte pouvant restituer les mêmes informations. Le but de la compression c'est de réduire le volume de données (le nombre de bits) nécessaire pour représenter et coder les caractéristiques d'une image (élimine la redondance d'information) ce qui permet de réduire le coût de stockage et d'avoir une transmission rapide des données.

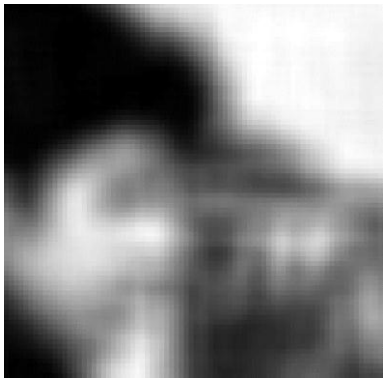
Dans le cadre de notre cours on s'intéressera à l'algorithme de compression **RLE** (Run-lenght encoding), appelé en français le codage par plages, consiste à indiquer pour chaque suite de pixels d'une même couleur, le nombre de pixels de cette séquence. Le résultat comporte en général moins de caractères. Plus le nombre de couleurs est important, moins nombreuses sont les chances de redondance. La méthode est donc intéressante pour les images à deux couleurs (noir et blanc).
Par exemple, la séquence « 0000FF000000FFFF » peut être encodée comme suit « 402F604F ».

Exercice 9.
1. Créer une fonction Python nommée **compression_RLE** qui prend en arguments un objet **img** de type *PIL.Image* et qui renvoie une liste **L** (de type *List* en Python) de tuples contenant les codes de pixels compressés selon la méthode RLE. Exemple : L = [(4,'0'),(2,'F'),(6,'0'),(4,'F')].
2. Créer une fonction Python nommée **decompression_RLE** qui prend en arguments un objet **L** de type *List* contenant les codes de pixels compressés selon la méthode RLE et qui affiche l'image créée (de type *PIL.Image* en Python) à partir de **L**.

2. Filtre moyenneur

Le filtre moyenneur est une opération de traitement d'images utilisée pour réduire le bruit dans une image et/ou flouter une image. Par exemple, l'application d'un filtre moyenneur sur l'image de gauche donne l'image de droite :

En zoomant, on peut voir en détail les effets du filtre; le bruit clairement visible dans le ciel a bien été réduit mais les détails du visage et de la caméra sont floutés :



Le filtre moyenneur fait partie de la catégorie des filtres d'images locaux car pour calculer la nouvelle valeur d'un pixel, il regarde la valeur des pixels proches. Concrètement, la valeur filtrée d'un pixel p est égale à la moyenne des valeurs des pixels proches de p. En général, on définit les « pixels proches de p » comme l'ensemble de pixels contenus dans un carré de largeur k centré sur p :

24	32	128	240	255
12	42	111	154	222
4	23	123	176	243
15	63	145	134	172
27	12	98	75	143

		108		

Avec un filtre moyenneur de paramètre k=3, pour calculer la nouvelle valeur du pixel rouge de l'image originale de gauche, on calcule la valeur moyenne des pixels situés dans un carré de dimension 3×3 centré sur ce pixel.

Cela donne la nouvelle valeur du pixel sur l'image transformée (pixel vert sur l'image de droite) :
42+111+154+23+123+176+63+145+1349=108

Cette opération est répétée pour tous les pixels de l'image. On parle de fenêtre glissante pour caractériser le carré sur lequel est calculé la moyenne des pixels et qui se déplace sur l'image. La fenêtre se déplace sur l'image du bas (en bleu) pour calculer les valeurs de la nouvelle image en haut (en vert).

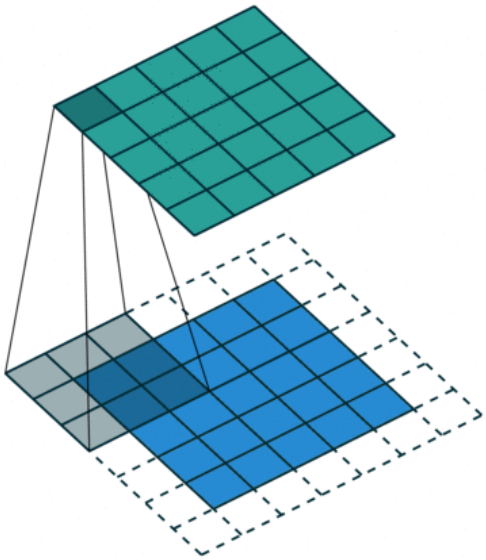
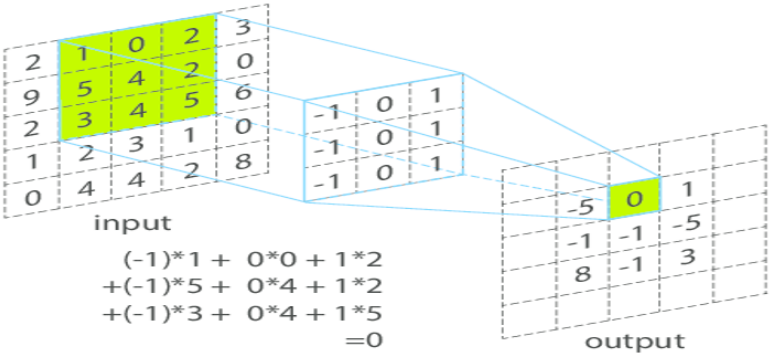


Illustration du principe de la fenêtre glissante.

Exercice 10.
Implémentez le filtre moyenneur dans la fonction **Filtre_moyenneur**.
Notez que pour une valeur de paramètre k, la fenêtre de calcul à utiliser est de taille 2k+1 : cette pratique commune permet de garantir que la fenêtre considérée est de dimension impaire et donc que son centre tombe précisément sur un pixel de coordonnées entières.

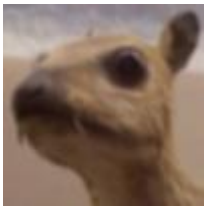
3. Convolution

La convolution, ou produit de convolution, est une généralisation du filtre moyenneur où l'on considère cette fois une moyenne pondérée. La fenêtre glissante est alors elle-même une image qui contient les coefficients de pondération. On l'appelle généralement noyau de convolution ou masque de convolution (kernel ou mask en anglais) :



Le noyau de convolution (au centre) contient les coefficients de pondération. Le principe est alors similaire au filtre moyenneur : pour calculer la nouvelle valeur d'un pixel à droite, on calcule la moyenne des pixels de l'image originale (à gauche) se trouvant sous le masque de convolution pondérée par les valeurs du masque.

Exemples :

Effet	Identité (ne fait rien…)	Filtre moyenneur (lissage)	Filtre gaussien 3×3 (lissage)	Filtre gaussien 5×5 (lissage)
Noyau	[1]	$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$	$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & 36 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}$
Résultat				

Autres filtres :
- Filtre Laplacien (détecteur de contours),
- Filtre réhausseur (renforce les contours), …

Exercice 11.

Le filtre de Prewitt utilise la matrice

$$A = \begin{bmatrix}-1 & -1 & -1\\0 & 0 & 0\\-1 & -1 & -1\end{bmatrix}$$

afin de définir le contour dans une image en niveaux de gris. Le principe consiste à calculer le produit de convolution pour chaque pixel de l'image originale par la matrice A comme l'illustre le schéma suivant :

p1	p2	p3	…
p4	p5	p6	
p7	p8	p9	
…			

-1	0	1
2	1	-2
0	2	1

			…
	S		
…			

Si on considère que le pixel p5 de l'image à gauche, le produit de convolution par la matrice A = [[-1, 0, 1], [2, 1, -2], [0, 2, 1]] est égal à : S = -1 × p1 + 0 × p2 + 1 × p3 + 2 × p4 + 1 × p5 + (-2) × p6 + 0 × p7 + 2 × p8 + 1 × p9
Et qui sera enregistrée dans la nouvelle image à droite à la même position que le pixel p5.

1. Ecrire une fonction convolution qui étant donné 9 pixels p1, p2, p3, p4, p5, p6, p7, p8 et p9 et une liste de listes A représentant une matrice 3×3, calcule le produit de convolution du pixel p5 comme le montre le schéma précédant.
2. Ecrire une fonction **appFiltreConv** qui prend en paramètre un tableau **T** à **nc** colonnes et à **nl** lignes stockant les pixels d'une image en niveaux de gris, un filtre **F** et une image résultat **img2** et qui applique la convolution sur tous les pixels de T (Attention au débordements).
3. Ecrire un script Python qui :
   a. Charge la classe Image du module PIL.
   b. Crée un objet img stockant l'image enregistrée dans : « c:\python32\image.jpg ».
   c. Convertit en une seule commande img en une image en niveaux de gris img1.
   d. Applique le filtre de Prewitt sur img1 afin de créer une image img2 contenant les contours de img1.
   e. Affiche img2.

4. Dilatation

Le dilaté d'un pixel est la valeur maximale au voisinage de ce pixel contenu dans l'image.
Exemple d'une dilatation sur une matrice de nombres :

12	4	63	156	2
84	42	21	84	84
232	14	24	8	9
1	3	161	94	5
9	66	8	96	33
8	5	169	8	48

12	4	63	156	2
84	42	21	84	84
232	14	161	8	9
1	3	161	94	5
9	66	8	96	33
8	5	169	8	48

Pour une image en niveau de gris, il s'agit d'éclaircir l'image.

Un autre problème intervient en programmation, c'est l'effet de bord. La gestion des bords en traitement d'images est assez importante car, dans la plupart des cas, on supprime les objets touchant les bords. Pour cela, nous allons agrandir l'image en ajoutant une bordure d'un pixel.

Exercice 12.

1. Ecrire une fonction **ourvrir_image** permettant d'ouvrir une image qui prend en arguments le nom d'image à ouvrir **img** et qui renvoie une matrice **M** qui la représente.

2. Ecrire une fonction **construire_image** qui prend en arguments une matrice **M** représentant les données de l'image à créer et un objet nommé **nom** de type chaine de caractères. Cette fonction permet de construire une image en niveaux de gris à partir de la matrice **M** et de la sauvegarder dans un fichier nommé **nom**. Dans le dossier courant. La fonction à définir ne renvoie rien.

3. Ecrire une fonction **bordure** qui prend en arguments une matrice **M** représentant les données d'une image et ne renvoie rien, permettant d'ajouter une bordure d'épaisseur **un** (1 pixel) à la matrice **M**.

4. Ecrire une fonction **dilatation** permettant d'appliquer l'algorithme de dilation.

Annexe : La classe Image du module PIL

```
>>> from PIL.Image import *
```

Méthodes	Signification
`img = open('tiger.jpg')`	Ouvrir un fichier image et créer un objet Image
`img.show ()`	Afficher l'image dans une fenêtre
`data = img.getdata()` `L = list(data)`	Retourner une séquence de valeurs des pixels d'une image
`Img_gris = img.convert('L')`	Retourner une copie d'une image convertie dans un mode différent. Les modes d'images en Python : ✓ « 1 » : Mode 1 bit ou mode Noir et Blanc ✓ « L » : Mode 8 bits ou mode Niveaux de gris ✓ « RGB » : Mode 24 bits ou mode RVB
`Img_gris.save('imgGris.bmp')`	Sauvegarder une image dans un fichier
`n=new('RGB',(Largeur,hauteur))`	Création d'un nouvel objet Image en précisant le mode et les dimensions Par défaut, tous les pixels sont mis à 0. l'image est noire.

`new('RGB',(100,100),(123,34,5))`
`new('RGB',(100,100),"red")`
`new('L',(100,100),123);`
`new('1',(100,100),"white")`
	Créer une image avec une couleur par défaut.

`nouveau.putdata(list(data))`	Mettre à jour les codes des pixels d'une image.
`p = Img_gris.getpixel((0,1))`	Récupérer la valeur d'un pixel à une position donnée
`Img_gris.putpixel((0,1),255)`	Modifier la valeur d'un pixel à une position donnée
`Img1= img.copy ()`	Créer une copie conforme d'une image
`img1.resize((120,120)).show()`	Créer une copie redimensionnée d'une image
`img.transpose(FLIP_LEFT_RIGHT).show()`	Créer une copie transposée d'une image
`Img_gris.rotate(45).show()`	Créer une copie de l'image originale déviée d'un angle de 45 degrés à partir de son centre dans le sens contraire des aiguilles d'une montre.
Attributs	**Signification**
`Largeur, hauteur = img.size` `print(Largeur, hauteur)`	Taille de l'image, en pixels. La taille est donnée sous la forme d'un tuple à 2 entiers (largeur, hauteur).
`img.mode`	Mode d'image. C'est une chaîne spécifiant le format de pixel utilisé par l'image. Les valeurs typiques sont : "1", "L", "RGB"
`img.width`	Largeur de l'image en pixels.
`img.height`	Hauteur de l'image, en pixels.

A retenir :

- `Image.open(path)` → `Image`
- `np.array(img)` → `np.array`
- `Image.fromarray(arr)` → `Image`