Épreuve du jeudi 24 février 2022 — durée : 3h

CONCOURS BLANC D' INFORMATIQUE (PYTHON ET SQL)

——— TRAITEMENT D'IMAGE POUR LA ROBOTIQUE EMBARQUÉE ———

Partie 1. Image : inspiré d'extraits de PSI 2011

On dote un robot d'exploration d'une camera afin de tester la possibilité de reconnaitre des obstacles. Le flux vidéo est découpé en une suite d'images qu'un traitement annexe transforme en images en **nuances de gris** (elles ne sont donc plus en couleurs). Une image est ici la donnée d'une hauteur H, d'une largeur L, et d'une matrice M d'entiers de H lignes et L colonnes. L'image est divisée en éléments ou pixels définis par leurs numéros de ligne noté ℓ et de colonne noté c. Chaque entier de la matrice définit le ton de gris associé au pixel de coordonnées (ℓ, c). Ce ton varie entre zéro, qui rend l'absence de lumière et donc le noir, et une valeur maximale dite profondeur P qui rend le blanc. Les valeurs intermédiaires rendent diverses teintes de gris de plus en plus claires. On notera que le pixel de coordonnées $(0, 0)$ est conventionnellement situé en haut et à gauche de l'image.

On suppose que l'on peut créer une image par l'appel de fonction `allouer(H, L, P)` . On accédera aux composants d'une image i par les notations `i.H` (hauteur), `i.L` (largeur), `i.P` (profondeur de gris) et `i.M` (matrice). On accédera aux éléments de la matrice par la notation `i.M[`ℓ, c`]`, sachant que les indices commencent à zéro, un indice de ligne est donc un entier ℓ compris entre 0 et $H - 1$ au sens large, tandis qu'un indice de colonne est un entier c compris entre 0 et $L - 1$ au sens large.

1 On veut écrire une fonction `inverser(i)` qui renvoie l'image inverse (inversion de contraste, par exemple un pixel blanc devient noir, et réciproquement) de l'image i. Complétez la ligne 9 ci-dessous.

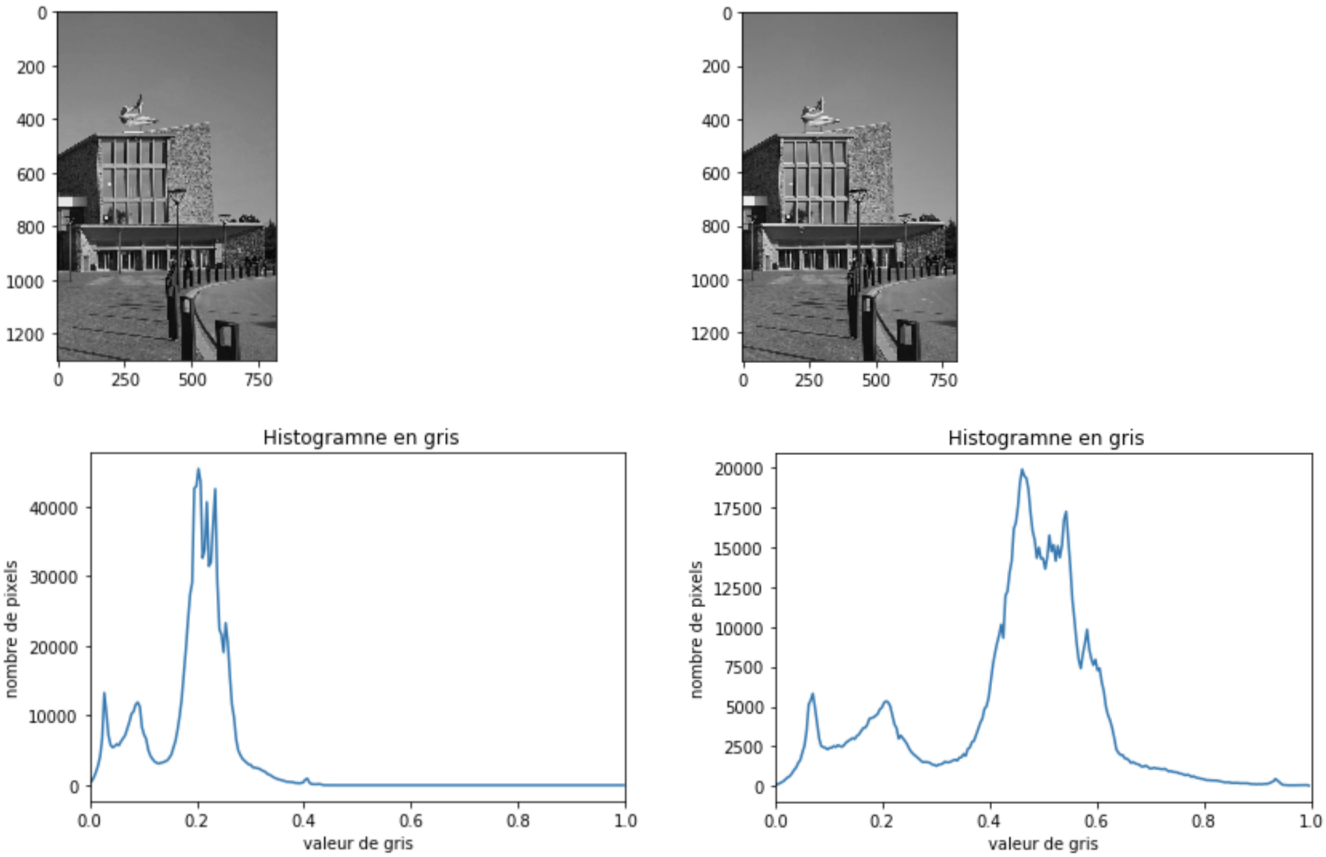
```
1  def inverser(i):
2  # Inverse les niveaux de gris d'une image
3          resultat = allouer(i.H,i.L,i.P)
4          t_original = i.M
5          t = resultat.M
6          profondeur = i.P
7          for j in range(i.H):
8                  for k in range(i.L):
9                   ... à compléter ...
10         return( resultat )
```

2 Écrire une fonction `juxtaposer(i_1, i_2)` qui prend en arguments deux images i_1 et i_2 de même largeur et profondeur, et qui renvoie la nouvelle image obtenue en juxtaposant i_1 et i_2. L'image i_1 est à gauche dans la nouvelle image, i_2 est à droite.

Vous utiliserez la trame suivante, en complétant les lignes 4 et 8 à 11 :

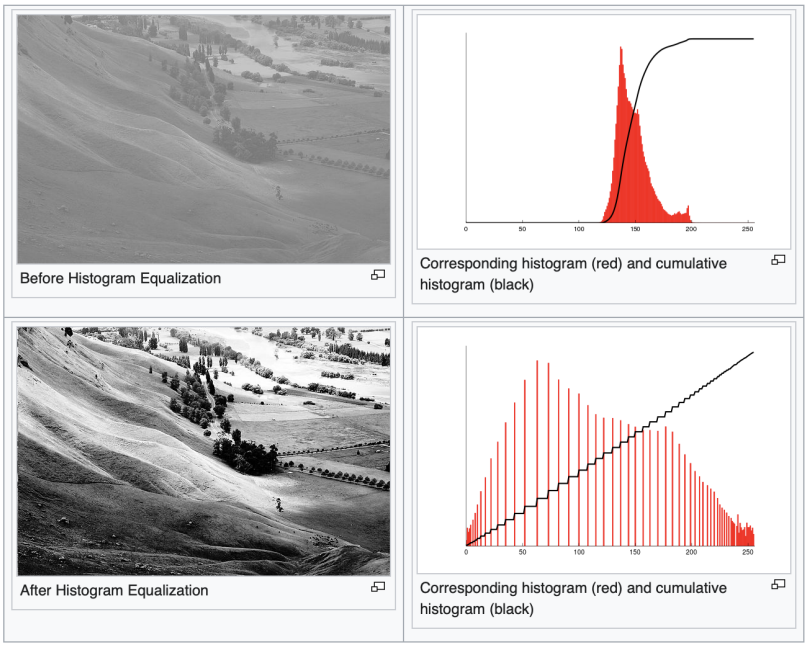
```
1  def justaposer(i1,i2):
2          l1 = i1.L    ,  l2 = i2.L
3          hauteur = i1.H
4          resultat = allouer(    ... à compléter ...    ,i1.P)
5          t1 = i1.M  ,  t2 = i2.M
6          t = resultat.M
7          for j in range(hauteur ):
8                  ...
9                  ...
10                 ...
11                 ...
12         return( resultat )
```

3 En traçant l'histogramme d'une image contrastée ou assombrie, on voit une diffférence dans le tracé, comme ci-dessous.



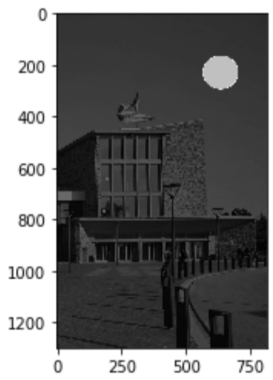
En abscisse, on a la valeur de gris notée entre 0 et P la profondeur, par exemple 256 pour 256 nuances de gris. En ordonnées, on a le nombre de pixels ayant cette valeur dans toute l'image. Quelle fonction mathématique semble cachée dans la différence de tracé des deux histogramme ?

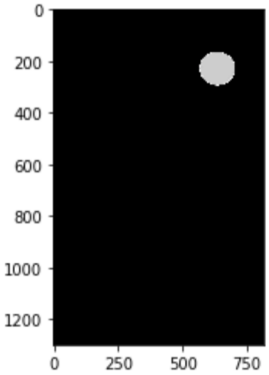
4 On cherche maintenant à effectuer le traitement allant d'une image peu contrastée à une image contrastée. Le cas ci-dessous correspond à une image sur-exposée, donc trop claire, avec peu de contraste dans les teintes sombres.

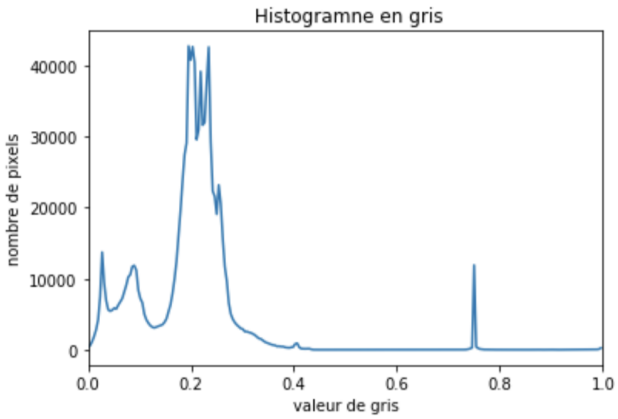


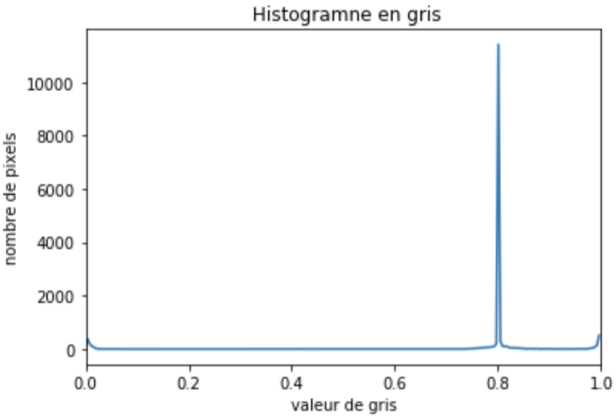
Quel est l'algorithme à appliquer pour passer d'un histogramme non contrasté à un histogramme contrasté réparti sur toutes les valeurs de l'axe horizontal ?

5 Un objet clair remarquable dont on veut suivre le mouvement passe dans le champ de la prise de vue. On veut isoler cet objet en le plaçant sur un fond noir.









Où est situé dans l'histogramme de gauche cet objet clair et uniforme ? Comment serait l'histogramme si l'objet n'était pas uniforme ?

6 Montrer que la fonction à réaliser ressemble à un filtre passe-haut dont vous préciserez la coupure.

7 On suppose que l'on récupère deux tableaux `pixels` et `gris` par l'appel de la fonction de bibliothèque `numpy` : `pixels, gris = np.histogram(image, bins=256, range=(0, 1))`
Comment modifier le tableau `pixels` pour isoler l'objet clair en remplaçant le fond par du noir ?

8 Pourquoi doit-on modifier `pixels[0]` avant l'appel du tracé ? Quelle valeur doit-on donner ?

Partie 2. Tri des listes de valeurs de gris

On veut maintenant savoir si une valeur de gris est utilisée pour un ou des pixels dans une image fournie.
On veut donc trouver si un élément recherché existe bien dans une liste. S'il existe, on fournit l'indice de la première occurrence trouvée. Sinon, il renvoie une réponse vide.

9 Comment passer d'un tableau de valeurs de gris comme précédemment à une liste de valeurs de gris dans l'ordre des lignes et colonnes ?

10 Dans une liste **triée**, on regarde la valeur centrale dans les indices encore disponibles. S'il ne s'agit pas de la bonne valeur, on enlève de l'intervalle de recherche les indices qui ne peuvent pas contenir la valeur recherchée. L'algorithme se décrit ainsi :

```
g = 0
d = longueur - 1
TANT QUE g <= d
c = (g + d) // 2
SI liste[c] < x
on ne garde que ce qui est à droite de l'index centre
g = (c + 1)
SINON SI liste[c] > x
on ne garde que ce qui est à gauche de l'index centre
d = (c - 1)
SINON
Si on arrive ici, c'est que x est à l'index centre
```

```
Renvoyer c
Fin du Si
Fin Tant que
Si on arrive ici, c'est que l'intervalle de recherche est vide
Renvoyer VIDE
```

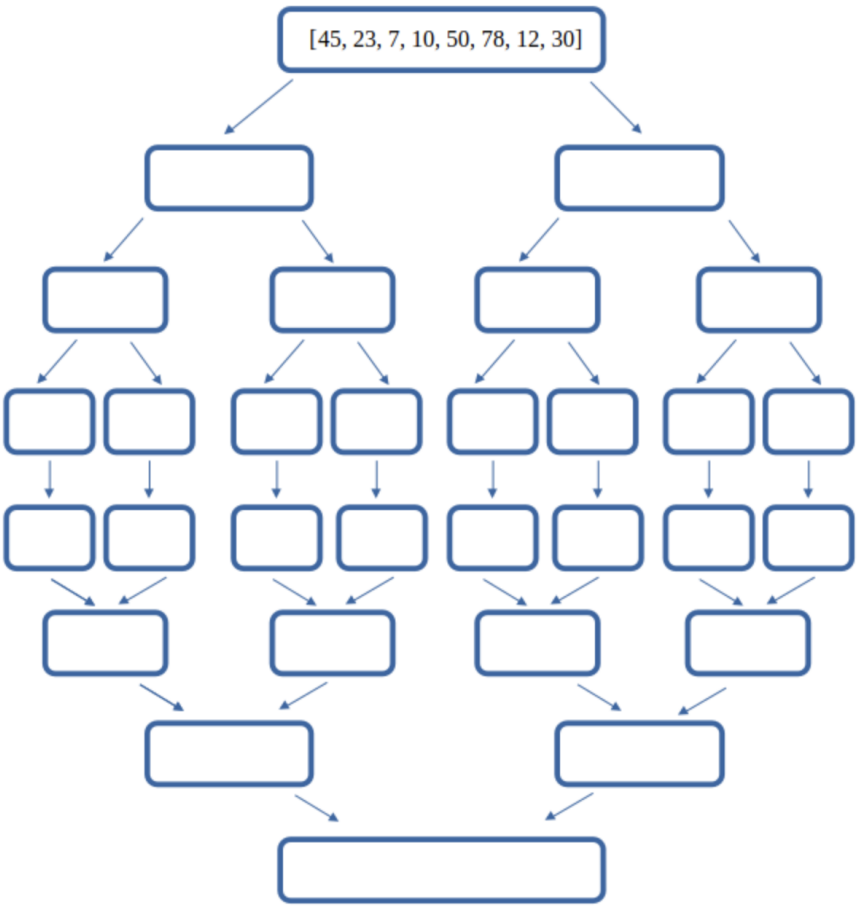
Commentez le terme de "recherce dichotomique" utilisé pour décrire cette méthode.

11 Nous allons donc utiliser le tri fusion qui consiste à :

— Phase DIVISER : on divise en deux notre tableau, récursivement.

— Phase REGNER : on règne sur le problème du tri lorsqu'on obtient... un tableau de une case. Elle est triée !

— Phase COMBINER : on fusionne alors facilement nos deux tableaux triés : le plus petit élément de chaque tableau est au début du tableau. Trier les éléments des deux tableaux revient donc à juste à chaque étape à lire deux valeurs et faire leur comparaison.



Remplir les cases de ce tri_fusion

12 Compléter le code de `tri_fusion()` suivant aux lignes 5 et 6 :

```
1  def tri_fusion(t):
2      n = len(t)
3      if n < 2:
4          return t
5      a =  ......
6      b =  ......
7      return fusion(a, b)
```

On utilisera le code de la fonction intermédiaire `fusion()`

```
1  def fusion(a, b):
2          p, q = len(a), len(b)
3          c = [None] * (p + q)
4          i=j=0
5          for k in range(p+q):
6                  if j >= q:
7                          c[k:] = a[i:]
8                          break
9                  elif i >= p:
10                         c[k:] = b[j:]
11                         break
12                 elif a[i] < b[j]:
13                         c[k] = a[i]
14                         i += 1
15                 else:
16                         c[k] = b[j]
17                         j += 1
18         return c
```

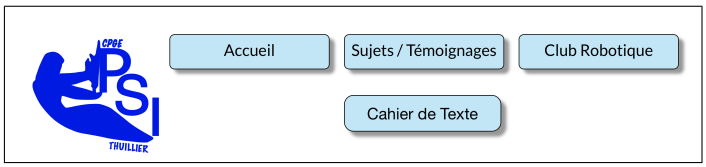
13 Quelle est la complexité de la fonction `fusion()` ?

Partie 3. Traitement des images envoyées par la camera

Nous allons reprendre le flux d'images extraites de la vidéo, mais cette fois nous utiliserons les images en couleurs. Nous aurons besoin des deux bibliothèques suivantes : `numpy` et `matplotlib.pyplot` :

```
1  import numpy as np
2  import matplotlib.pyplot as plt
```

La seconde contient deux fonctions qui vont nous être utiles :
- la fonction `plt.imread` prend en argument une chaîne de caractère décrivant le chemin d'accès à un fichier image au format PNG par exemple et retourne un tableau NUMPY. Ce tableau a même dimension que l'image, et chacune de ses cases contient un triplet donnant une liste (triplet) des composantes RGB du pixel correspondant ;
- la fonction `plt.imshow` prend en argument un tableau NUMPY et affiche l'image associée à cette matrice, éventuellement suivie de l'instruction `plt.show ( )` si le mode interactif n'est pas activé.

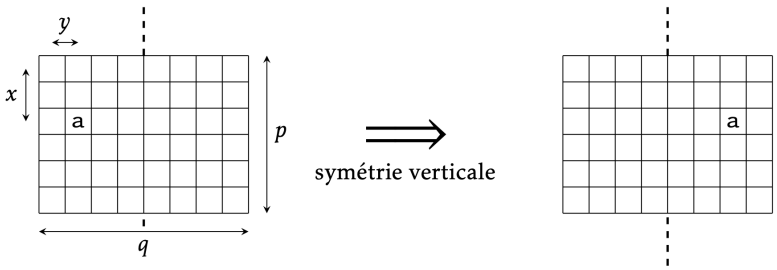


On affiche l'image ci-dessus grâce au code suivant :

```
1  site = plt.imread('sitepsi.png')
2  plt.imshow(site)
```

Les dimensions $p \times q$ de l'image initiale peuvent être calculées par les instructions `p=img.shape[0]` et `q=img.shape[1]`. Ici, pour l'image ci-dessus, on a 578×2824

14 Dans le cas d'une symétrie d'axe vertical, l'image transformée a les mêmes dimensions que l'image initiale. Pour créer la matrice-image vierge, on utilisera l'instruction `np.zeros_idem(img)` qui prend en argument une matrice `img` et qui renvoie une matrice vierge de mêmes dimensions.



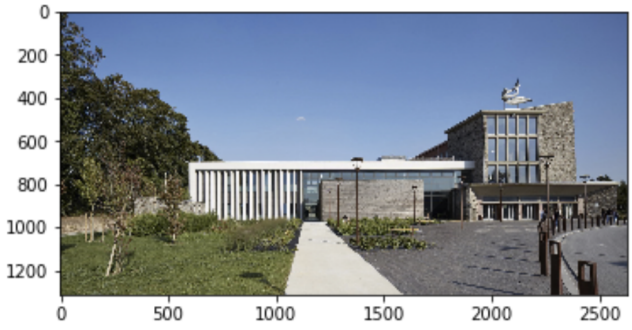
Pour des besoins de symétrie d'image lorsque l'on donne l'ordre au robot d'inverser son sens de déplacement, étant donné un pixel `a` de coordonnées (x, y) dans l'image initiale, comme représenté sur le schéma précédent, exprimer ses coordonnées (x', y') dans l'image résultat d'une symétrie d'axe vertical passant par le centre de l'image.

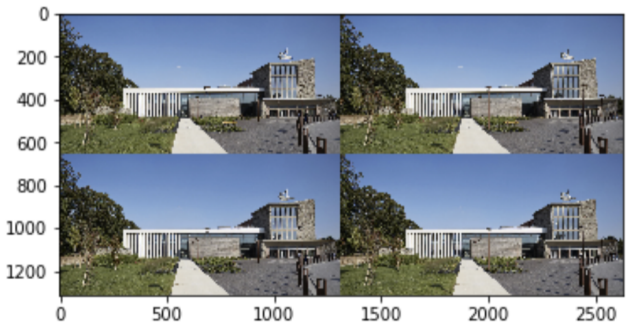
15 On va construire une fonction `symetrie(img)` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la symétrie d'axe vertical. Les dimensions $p \times q$ de l'image initiale peuvent être calculées par les instructions `p=img.shape[0]` et `q=img.shape[1]`. Complétez le code suivant aux lignes 4 et 5 en utilisant le fait que `img[x, y]` correspond à une sous-liste `[r,v,b]` des 3 couleurs RVB de ce pixel à la ligne x et la colonne y de ce tableau.

```
0  def symetrie(img):
1          p, q = img.shape[0], img.shape[1]
2          img2 = np.zeros_idem(img)
3          for x in range(p):
4
5
6          return img2
```

16 Comment coder la fonction `np.zeros_idem(img)` ? Pourquoi la ligne 1 du code précédent semble redondante ?

17 Comme le contrôleur humain du robot a un retour vidéo de la camera embarquée, il est intéressant de visualiser sur l'écran les différents résultats de traitements d'image : on veut placer les différentes images modifiées par filtrage et traitement (par exemple 3) en plus de l'image initiale sur les 4 parties de l'écran, divisée en zones égales. C'est ensuite sur les quarts d'écran que seront effectués les filtrages et traitements, comme par exemple la reconnaissance d'objets. Montrons le principe avec pour le moment la copie de quatre fois la même image en couleurs de dimension 1312×2632.

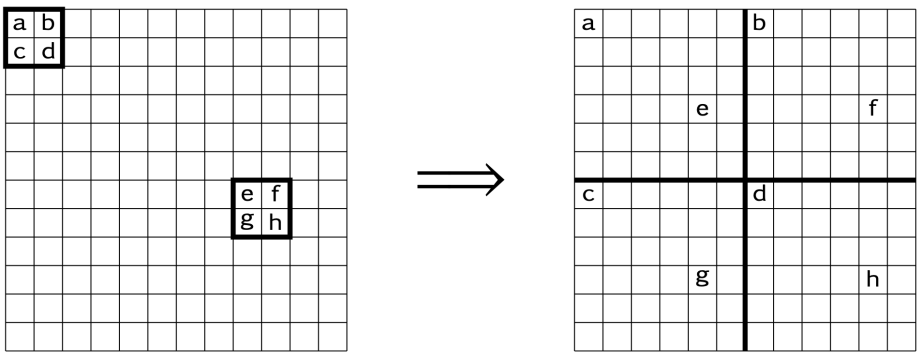




On a une **contrainte** : comme vous le voyez sur le quadrillage des images précédentes, l'image finale a la même taille que l'image initiale. Il ne s'agit donc pas de recopier (fonction `crop()` en Python) l'image initiale quatre fois, sinon on obtient une image quatre fois plus grande, ce qui encombre la mémoire. Une méthode serait donc de faire la moyenne d'un paquet de 4 pixels sur le dessin de gauche (voir le schéma `abcd` en dessous) et de le placer aux positions des pixels notés a, b, c et d du dessin de droite. On ne choisit pas cette méthode.

Pour que cette transformation soit bijective, on a découpé l'image initiale en paquets carrés de quatre pixels (2×2) puis pour chaque paquet carré de quatre pixels, on utilise celui en haut à gauche pour l'image réduite en haut à gauche, celui en haut à droite pour l'image réduite en haut à droite, etc.

On notera que pour que cette transformation soit facilement définie il est nécessaire que les dimensions horizontale et verticale de l'image soient paires et qu'une image carrée $2n \times 2n$ serait idéale à traiter.



La méthode utilise une transformation qui utilise les formules :

$$(x', y') = \begin{cases} (x/2, y/2) & \text{si } x \text{ et } y \text{ sont pairs} \\ (x/2, y/2 + q/2) & \text{si } x \text{ est pair et } y \text{ impair} \\ (x/2 + p/2, y/2) & \text{si } x \text{ est impair et } y \text{ pair} \\ \textbf{... à compléter ...} & \text{si } x \text{ et } y \text{ sont impairs} \end{cases}$$

On rappelle que la division entière est notée `//` et que le reste est noté `%`. Ainsi, `5 // 2 = 2` et `5 % 2 = 1`. En quoi l'écriture de la fonction suivante `distribution` plus concise, à deux tests au lieu de 4, permet de traduire la transformation (x', y') ci-dessus ? Complétez la ligne 4.

```
0 def distribution(k, d):
1         if k % 2 == 0:
2                 return k // 2
3         else:
4                 return  ... à compléter ...
```

18 Dans le code précédent, on utilise la division entière "//" au lieu de la division décimale "/". Pourquoi ?

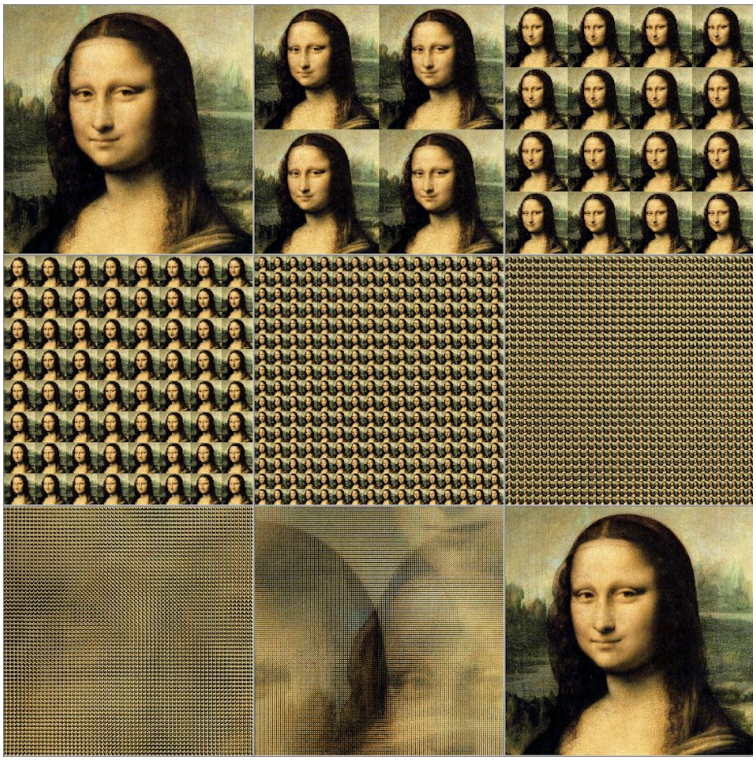
19 Si, comme dans certains autres langages, le Python ne possédait pas cette instruction "//", comment pourrait-on la réaliser avec la fonction `int()`. Rappelons que `int(3.6)` donne `3`. Comment pourrait-on obtenir l'opérateur `%` en écrivant une fonction `reste(a,b)` basée sur certaines des 4 seules opérations + - * / et également la fonction `int()` ?

20 Si l'on ne veut pas utiliser `int()` dans `reste(a,b)`, permettant de déterminer le reste de la division euclidienne d'un entier naturel a par un entier naturel non nul b, on peut utiliser l'algorithme simpliste :
entrées : a et b entiers naturels, avec $b \neq 0$
initialiser une variable q à 0
initialiser une variable r à a
tant que r $\geqslant b$:
  remplacer q par q+1
  remplacer r par r $- b$
sortie : la valeur finale de r
Traduire cet algorithme en fonction `reste2(a,b)` dans le langage Python.

21 On va maintenant utiliser la fonction `distribution` pour obtenir un tableau correspondant à l'image composée par le critère (x', y'). Complétez la ligne 5 de la fonction `quatre_vues` ci-dessous.

```
0 def quatre_vues(img):
1         p, q = img.shape[0], img.shape[1]
2         img2 = np.zeros_idem(img)
3         for x in range(p):
4                 for y in range(q):
5                         img2[  ...   ,    ...   ] = img[x, y]
6         return img2
```

22 Il existe une particularité non abordée pour cette méthode que nous venons de développer : des appels successifs de la fonction `quatre_vues` sur l'image précédemment donnée par cette même fonction donnent une évolution de l'image perçue comme sur le cas ci-dessous, avec comme image initiale une partie carrée $2n \times 2n$ du tableau de la Joconde.



Au bout d'un certain nombre d'appels itératifs de `quatre_vues`, l'image de départ ré-apparait clairement. Ce nombre d'appels nécessaires dépend de quel(s) paramètre(s) ?

23 Peut-on envisager d'utiliser ces observations pour cacher un texte dans l'image de la Joconde, en supposant qu'il existe une fonction `ajout(texte, coordonnees [x,y], image)` ? Comment procéder pour utiliser `ajout()` ?
Données : `coordonnees [x,y]` correspond à la position dans l'image du texte à partir du pixel `[x,y]` qui est la partie en haut à gauche de l'image équivalente au texte et `image` est l'image sur laquelle porte cet ajout de texte. Le retour de `ajout()` est un tableau image.

Partie 4. Codage binaire à améliorer en vue de transmissions sans erreurs

Le codage binaire (avec 0 et 1) doit parfois être consolidé. Avec deux niveaux de tension, l'échange des données d'une image sous la forme d'une suite de nombres binaires peut comporter des erreurs de transmission. Des algorithmes de détection ou correction d'erreur existent.
Il a été testé une **autre méthode** : le codage non pas en base 2 mais EN BASE 3.
En base 3 classique, on écrit les nombres à l'aide de trois chiffres : 0,1 et 2 : par exemple, $23 = \overline{212}^3 = 2\times 3^2 + 1\times 3 + 2$ ou encore $46 = \overline{1201}^3 = 1\times 3^3 + 2\times 3^2 + 0\times 3 + 1$.
En électronique, les tensions nécessaires pour le 0 et le 1 informatique existent déjà. Afin de ne pas modifier trop les sources de tensions, on préfère cette base 3 :
en base 3 modifiée, les "chiffres" utilisés sont $0, 1$ et -1, et on notera m la base :

$$23 = \overline{10(-1)(-1)}^m = 1\times 3^3 + 0\times 3^2 - 1\times 3 - 1$$
$$46 = \overline{1(-1)(-1)01}^m = 1\times 3^4 - 1\times 3^3 - 1\times 3^2 + 0\times 3 + 1$$

Dans toute la suite, on s'intéresse à l'écriture en base 3 modifiée des entiers naturels non nuls. On note $C = \{-1, 0, 1\}$ l'ensemble des chiffres utilisés dans l'écriture en base 3 modifiée.
L'écriture en base 3 modifiée d'un entier naturel non nul sera notée comme ci-dessus, par la suite des chiffres surmontée d'un trait avec un exposant e. Si $a_0, a_1, \ldots, a_{p-1}$ sont dans C, on a donc :

$$\overline{a_0 a_1 \ldots a_{p_1}}^m = \sum_{k=0}^{p-1} a_k 3^{p-1-k}$$

L'écriture en base 3 modifiée d'un entier naturel non nul, lue de gauche à droite, commence toujours par le chiffre 1.

24 Donner la valeur de $\overline{1(-0)0(-1)}^m$, de $\overline{1111}^m$ et de $\overline{1(-1)(-1)(-1)(-1)}^m$

25 Écrire en base 3 modifiée les entiers de 1 à 9.

26 Soit k un entier naturel. Déterminer la valeur de $A_k = \overline{1\ \underbrace{11\ldots1}_{k \text{ chiffres}}}^m$ et de $B_k = \overline{1(-1)(-1)\ldots(-1)}^m_{k \text{ chiffres}}$ (On aura bien noté que ces deux écritures possèdent $k+1$ chiffres au total.)

27 On représente l'écriture en base 3 modifiée $\overline{1(-1)(-1)(-1)(-1)}^m$ par la liste Python $[a_{p-1}a_{p-2},\dots,a_0]$ (attention, les chiffres sont écrits dans la liste en lisant de droite à gauche l'écriture en base 3 modifiée, le dernier d'entre eux est donc toujours le chiffre 1). **On veut écrire** à l'aide du code ci-dessous une fonction `valeur`(L) qui prend en argument une liste de chiffres et renvoie l'entier naturel non nul dont c'est une écriture en base 3 modifiée.
Par exemple `valeur`$([1,0,-1,-1,1])$ renvoie l'entier 46.
Vous allez vous aider de la fonction Python `enumerate` qui fonctionne de la manière suivante : le code suivant

```
0 for compteur, valeur in enumerate([1,0,-1]):
1         print(compteur, valeur)
```

donne une sortie 0 1
1 0
2 -1
On va donc implémenter la fonction demandée à partir de la trame suivante :

```
0 def valeur(L):
1 return sum(a * 3 ** index for            in enumerate(L))
```

Expliquer ce que doit contenir la boucle `for`
28 On veut montrer que tout entier naturel non nul n admet une écriture en base 3 modifiée. Soit $n\in\mathbb{N}^*$. On note q et r le quotient et le reste dans la division euclidienne de n par 3 : $n=3q+r$ et $r\in\{0,1,2\}$. On suppose que q est non nul et admet une écriture en base 3 modifiée $q=\overline{a_0\dots a_{p-1}}^m$. Déterminer une écriture en base 3 modifiée de n dans le cas où $r=0$ ou $r=1$.
29 On suppose que q est non nul et que $q+1$ admet une écriture en base 3 modifiée $q+1=\overline{b_0\dots b_{p-1}}^m$. Déterminer une écriture en base 3 modifiée de n dans le cas où $r=2$.
30 Montrer par récurrence que tout entier naturel non nul admet une écriture en base 3 modifiée.
31 On souhaite écrire en Python une fonction `incremente(L)` qui prend en argument une liste L de chiffres représentant une écriture en base 3 modifiée d'un entier naturel non nul n et qui renvoie une liste représentant une écriture en base 3 modifiée de $n+1$.
On propose la fonction récursive `incrementeR(L)` suivante. Expliquer les lignes 7 et 8. Pourquoi a-t-il fallu écrire les lignes 1 et 2 ?

```
0 def incrementeR(L) :
1         if len (L)==0 :
2                 return [1]
3         elif L[0]==0 :
4                 return [1]+L[1:]
5         elif L[0]== -1 :
6                 return [0]+L[1:]
7         else :
8         return [-1] + incrementeR(L[1:])
```

32 On veut réaliser une version itérative de la fonction `incremente(L)` . Expliquer ce que l'on doit coder en lignes 13 et 14 dans le code ci-dessous.

```
0  def incremente(L):
1          p = len(L)
2          M = []
3          k = 0
4          while k < p and L[k] == 1:
5                  M.append(-1)
6                  k += 1
7                  if k == p:
8                          M.append(-1)
9                  elif L[k] == 0:
10                         M.append(1)
11                         M = M + L[k+1:]
12                 elif L[k] == -1:
13
14
15         return M
```

33 Pour un résultat de type `[1,0,-1,1,0,0,-1,1,1]`, correspondant au nombre N, pouvez vous prédire le résultat de `valeur(-N)` ?

34 On souhaite montrer que l'écriture en base 3 modifiée d'un entier naturel n non nul est unique. Soit $a_0, \dots, a_{p-1}$ des chiffres de C (avec $a_0 = 1$ et $p \geqslant 1$). Quel est le reste dans la division euclidienne par 3 du nombre $\overline{a_0 \dots a_{p-1}}^m$?

35 En déduire que l'écriture en base 3 modifiée d'un entier naturel n non nul est unique.

36 On veut écrire une fonction `base3(n)` qui prend en argument un entier naturel n non nul et renvoie la liste L de chiffres qui correspond à l'écriture en base 3 modifiée de n.
Par exemple `base3(46)` renvoie $[1, 0, -1, -1, 1]$.
Vous compléterez le code ci-dessous en lignes 6 et 7. A quoi sert la ligne 2 avec `ajout` ?

```
0  def base3(n):
1          chiffres = [0,1,-1]
2          ajout = [0, 0, 1]
3          L = []
4          while n > 0:
5                  reste = n % 3
6                  L.append(        )
7                  n =
8          return L
```

— FIN —