Projet 3d - Qr_code (3444-3798269)

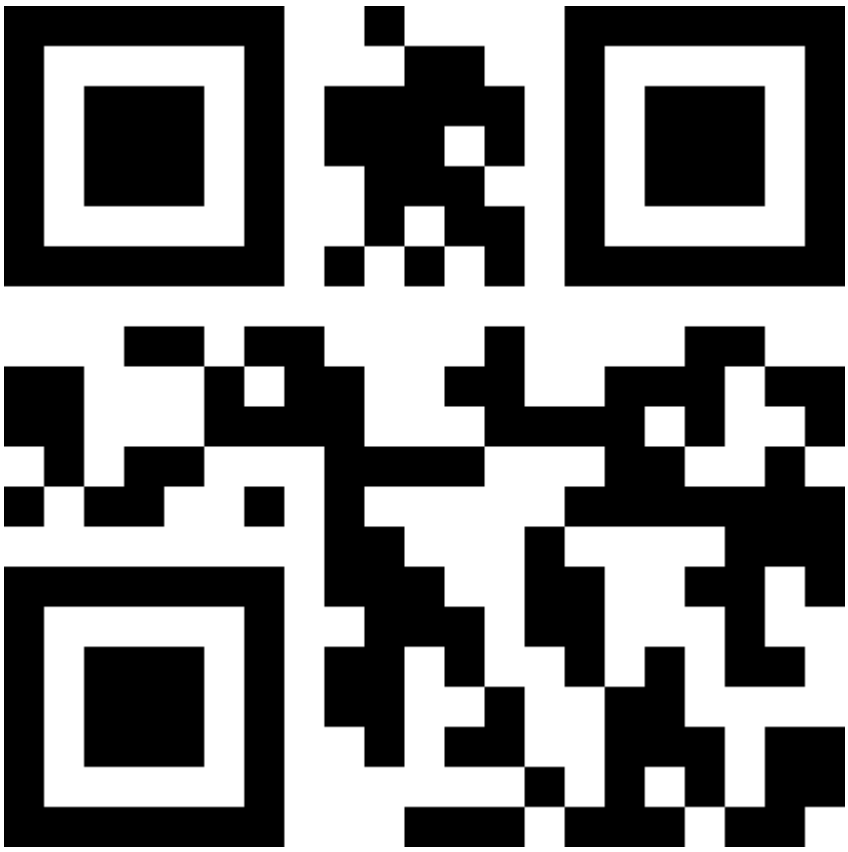
Un QR code est un tableau de $420 * 420$ pixels. Ici on considère une pixel en niveau de noir et blanc. Un pixel a donc pour valeur 0 ou 1.

Ce tableau est en fait consitué de modules. Un module est ensemble de pixels identiques de dimension $20 * 20$.

Le tableau total est alors constitué de $21 * 21$ modules de ce type.

Dans ce tableau de pixels regroupés en module on retrouve :

- de motifs de positionnement (3 blocs) situés aux coins permettant d'aider à la lecture du code
- des motifs de format d'information



1) Écrire une fonction *init* qui réalise l'initialisation d'une liste de dimension n où chaque élément est également une liste de dimension n. Cette liste de listes représente ainsi une matrice de taille $n * n$. Cette fonction a un paramètre de type int et retourne une liste de listes qui représente un QR code initialisé avec des valeurs 0.

In [2]:
```
#exemple
init(4)
```

On donne ci-dessous les lignes de code qui permettront d'avoir une liste $420 * 420$, appelée *liste*, associé à un QR code et qui permettra de tester vos programmes

In [7]:
```
from PIL import Image
import numpy as np
#en ligne, on peut générer facilement des QRcode
#https://www.nayuki.io/page/qr-code-generator-library
im=Image.open("C:/Users/User/Documents/travail/TSI2_2024_2025/infor
tab=np.array(im,dtype="uint8")
tab=tab[:,:,0]//255#quadruplet initial
image_new=Image.fromarray(tab)
liste=tab.tolist()#liste de liste de triplets de dimension (420*420

```

2) Écrire la fonction *chargevaleur* qui a pour but de réduire les données de l'image dans une liste de listes de dimension $21 * 21$. On rappelle que l'image initial correspondant à un QR code représente une liste de listes de dimension $420 * 420$ pixels dont on veut réduire tous les blocs constitués de $20 * 20$ pixels à un seul pixel pour avoir à partir de l'image une liste de listes de dimension $21 * 21$. Ne pas oublier que tous les pixels d'un bloc sont identiques. Cette fonction a un paramètre de type image et retourne une liste de listes de triplets (couleur des pixels).

Dans la suite, on travaillera avec cette nouvelle liste réduite, appelée *listereduite* de dimension $21 * 21$ associée au QR code

In [2]:
```
listereduite=[[0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0
               [0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1
               [0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0
               [0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0
               [0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0
               [0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1
               [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0
               [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
               [1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0
               [0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0
               [0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0
               [1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1
               [0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0
               [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1
               [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0
               [0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1
               [0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1
               [0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1
               [0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0
               [0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0
               [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1
```

3) On considère un bloc de positionnement comme donné ci-dessous. Écrire une fonction cree_bloc qui crée un bloc de positionnement. Cette fonction, qui n'a pas de paramètre, retourne une liste de listes de dimension 7*7.

In [31]:
```
1 # le programme doit renvoyer la liste ci-dessous
```

Out[31]:
```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 1, 1, 1, 1, 1, 0],
 [0, 1, 0, 0, 0, 1, 0],
 [0, 1, 0, 0, 0, 1, 0],
 [0, 1, 0, 0, 0, 1, 0],
 [0, 1, 1, 1, 1, 1, 0],
 [0, 0, 0, 0, 0, 0, 0]]
```

4)Écrire une fonction test_bloc qui teste si un bloc de positionnement (rappel : il y en a trois) est bien représenté pixel par pixel dans un QR code. Cette fonction a 3 paramètres : les coordonnées x et y donnant la position du début d'un bloc de positionnement d'un QR code (toujours les coordonnées du pixel le plus haut et à gauche) et la liste de listes de dimension $21 * 21$ associée au même QR code. Cette fonction retourne un booléen.

Remarque : on cherche à tester si un bloc de positionnement d'un QR code n'a pas subi une modification. Les coordonnées du pixel le plus haut et à gauche pour le premier bloc sont égales à (0,0), pour le second bloc à (0,14) et pour le troisième bloc à (14,0). Exemples :

test_bloc(0,0, mat1) =>renvoie True

test_bloc(1,3, mat1) =>renvoie False

5) On considère qu'un QR code est bien positionné lorsque ses 3 blocs de contrôle sont effectivement présents en haut à gauche, en haut à droite et en bas à gauche (comme présenté au début du sujet). Écrire une fonction test_QRcode qui permet de tester si un QR code est bien positionné. Cette fonction a pour paramètre une matrice de dimension 21*21 et retourne un booléen.

Exemple :

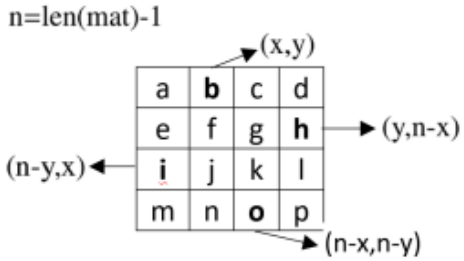
test_QRcode(mat1)=> renvoie True

Lors de la lecture d'un QR code par un appareil dédié (scanner, caméra ou autre) le processus de lecture permet de placer un QR code dans l'une des quatre positions possibles, comme illustré ci-dessous. Cela dépend bien évidemment de l'orientation du QR code lors de sa lecture.

On se propose de faire tourner un QR Code par rotation successive de 90° afin qu'il puisse se trouver dans la bonne position comme celui de la 1e figure (au début du sujet).

Pour construire ce programme, on se limite à un exemple d'une liste de listes de dimension $4 * 4$ pour expliquer le fonctionnement, mais ce serait la même chose pour une liste de listes de dimension *21*21*.

Si on prend les 4 éléments (b, h, o, i) de la liste de listes table 1 de la figure ci-dessous, b doit se trouver, après une rotation de 90°, à la place de l'élément h, l'élément h à la place de l'élément o, l'élément o à la place de l'élément i et l'élément i à la place de l'élément b. Le résultat de la transformation est illustré dans la table 2. Le mécanisme doit s'exécuter de la même manière sur les autres éléments de la table 2. Le résultat de la transformation finale est illustré dans la table 3.



a	**b**	c	d
e	f	g	**h**
i	j	k	l
m	n	**o**	p

Table 1

a	**i**	c	d
e	f	g	**b**
o	j	k	l
m	n	**h**	p

Table 2

m	i	e	a
n	j	f	b
o	k	g	c
p	l	h	d

Table 3

6) Écrire une fonction *tourHoraire* qui réalise une rotation de 90°, dans le sens des aiguilles d'une montre, à 4 pixels du QR code. La fonction a trois paramètres, les coordonnées x et y d'un élément de la liste de listes et une liste de listes de dimension 21*21.

7) Écrire une fonction *rotationHoraire* qui réalise la rotation de 90° d'un QR code. Cette procédure a un seul paramètre, une liste de listes de dimension 21*21.

8)Connaissant les 4 positions possibles lors de la lecture d'un QR code par un appareil dédié, écrire la procédure QRcode_posi qui positionne correctement un QR code. Cette procédure a un seul paramètre, une liste de listes de dimension 21*21.

Indication : utiliser les fonctions rotationHoraire et test_QRcode .