Chiffrement d'un texte dans une image

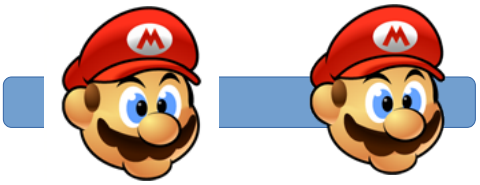
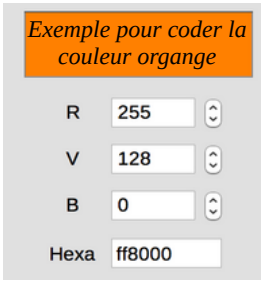


image .jpg image .png

Le principe est de modifier de façon non perceptible le codage des couleurs d'un pixel pour y loger, pixel après pixel, une partie de l'information correspondant au codage d'une lettre d'un message.

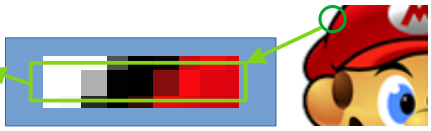
Niveaux R-V-B (-A)



Chaque pixel de couleur est constitué de trois niveaux de luminosité pour chaque couleur primaire Rouge, Vert et Bleu. *Pour les images de type png, il existe un quatrième indicateur : la transparence du pixel par rapport au fond (couche Alpha)*

Le principe de la synthèse additive permet donc de coder 256*256*256 couleurs différentes par pixel.

8 Pixels								
Rouge	255	255	125	20	20	127	241	200
Vert	255	255	125	20	20	10	13	26
Bleu	255	255	125	20	20	10	13	26



Pour loger l'information correspondant à nos caractères dans l'image, on va choisir de ne modifier par exemple que le niveau de bleu. On peut représenter **ce niveau de bleu sous forme binaire** :

Bleu	255	255	125	20	20	10	13	26
	1111 111**1**	1111 111**1**	0111 110**1**	0001 010**0**	0001 010**0**	0000 101**0**	0000 110**1**	0001 101**0**

Prenons par exemple la lettre 'z'.
Son code ASCII dans la table des caractères est 122. La valeur 122 se code `0 1 1 1 1 0 1 0` en binaire.
On va donc modifier les bits « de poids faible » de ces huit pixels ci-dessus pour y loger **un bit du code binaire de notre lettre 'z' dans chacun des pixels :**

	1111 111**0**	1111 111**1**	0111 110**1**	0001 010**1**	0001 010**1**	0000 101**0**	0000 110**1**	0001 101**0**
Nouveau niv. Bleu	254	255	125	21	21	10	13	26

Ces faibles modifications ne seront pas perceptibles dans l'image. Dans une image de 128 × 128 pixels, on dispose donc de 16 384 pixels. Chaque caractère de notre texte est codé sur 8 bits. Avec 1 bit d'un caractère par pixel, on peut donc dissimuler dans l'image 16384 ÷ 8 soit 2048 caractères !!!

Algorithme :

Coder une fonction qui reçoit le caractère à coder et retourne son code ASCII sous forme d'une liste de bits.

Ouvrir et récupérer la définition de l'image.

Pour chaque caractère du message :
    # Convertir le caractère en liste de bits
    # Pour chaque bit du caractère :
        # Lire les valeurs RVB d'un pixel
        # Remplacer le bit de poids faible de la composante Bleu avec le bit par le bit du caractère
        # Replacer le pixel dans l'image
        # Passer au pixel suivant
Enregistrer l'image modifiée.

Pour décoder l’image et retrouver notre message, il s’agit d’effectuer les opérations dans le sens inverse.

Coder une fonction qui reçoit une liste de bits (code ASCII) et retourne le caractère correspondant

Tant que le message n’est pas terminé et qu’il reste des pixels :
    # Lire la composante R,V,B d’un pixel
    # Mémoriser le bit de poids faible dans une liste
    # Si la liste contient 8 bits :
    `   # Convertir les huit bits en caractère
        # Mémoriser le caractère
    # Passer au pixel suivant
Afficher le message

L’image qui sert de support est située dans le même dossier de le script Python

```
""" Modifier les pixels de l'image """
from PIL import Image

def code_binaire(un_caractere:str)->list:
    code = ord(un_caractere) # récupérer le code ascii
    code_char = list(bin(code)) # le transformer en binaire
    for _ in range(2) : # retirer les deux premiers éléments
        code_char.pop(0)
    code = [ int(i) for i in code_char ]
    for _ in range(8-len(code)):
        code.insert(0, 0) # compléter pour obtenir 8 bits
    return code

message="""Bonjour,
    ce texte va être dispersé dans les valeurs de bleu
    de différents pixels de l'image ..."""

message = message + ".FIN" # Marqueur de fin de message

img = Image.open("mario-128x128.bmp")
# ========================== Lire des informations sur l'image
nb_lgn, nb_col = img.size

lgn = 0
col = 0
if len(message) < (nb_lgn*nb_col)//8 :
    for un_caractere in message :
        liste =
        for un_bit in code_binaire(un_caractere): # bit poids fort -> faible

            R, V, B = img.getpixel( (lgn, col) )

            B = B & 0xFE # mettre le bit de poids faible à 0
            B = B | un_bit # bit du message dans la composante bleue du pixel

            img.putpixel ( (lgn, col), (R, V, B) )

            col = col+1 if col < nb_col-1 else 0 # pixel suivant
            if not col :
                lgn += 1
else :
    print("Définition de l'image insuffisante pour stocker ce message !")

img.save("mario_message.bmp")
img.close()
```

```
""" Récupérer le message """
from PIL import Image

def binaire_caractere (liste_bits:list)->str:
    code_ascii = 0
    for i in range(8):
        code_ascii += liste_bits[i] << i
    return chr(code_ascii)

message=""

img = Image.open("mario_message.bmp")
# ================ Lire des informations sur l'image
nb_lgn, nb_col = img.size

liste_bits = []
lgn=0
col=0
while not ".FIN" in message and lgn<nb_lgn:

    R, V, B = img.getpixel( (lgn, col) )

    liste_bits.insert(0,(B & 1)) # Placer le bit de poids faible dans la liste

    if len(liste_bits)==8 :
        caractere = binaire_caractere ( liste_bits )
        liste_bits=[]
        message += caractere

    col = col+1 if col<nb_col-1 else 0 # Passer au pixel suivant
    if not col :
        lgn += 1

img.close()

message = message[:len(message)-4]  # Retirer le marqueur de fin ".FIN"
print(message)
```

======= Pour information : =======

```
# pour visualiser l'image :
img = Image.open("mario_message.bmp")
img.show()
img.close()
```