Devoir maison

L'objectif de ce devoir est d'étudier une technique rudimentaire de stéganographie permettant de cacher un message dans une image

Ce devoir tient lieu de contrôle continu

Vous pouvez le faire en binôme ou monôme. Vous rendrez votre devoir au plus tard le **1er avril** via ecampus. Vous déposerez une archive contenant votre fichier steganographie.ipynb et les images que vous utilisez pour vos tests. Un seul devoir est à déposer par binôme.

Question 0. Indiquez vos noms, prénoms et numéros d'étudiants
Nom :
Prénom :
Numéro d'étudiant :

Nom :
Prénom :
Numéro d'étudiant :

Partie 1 - Introduction

On va utiliser des images couleur au format RGB (Red, Green, Blue). La couleur de chaque pixel de l'image est ainsi codée par un vecteur de 3 entiers compris entre 0 et 255. Le premier entier donne la quantité de rouge, le deuxième la quantité de vert et le troisième la quantité de bleu. La synthèse de ces trois quantités détermine alors la couleur du pixel.
On rappelle qu'il suffit de 8 bits (un octet) pour coder en binaire tous les entiers compris entre 0 et $255 = 2^8 - 1$.

Le principe général
L'idée de départ est qu'on peut modifier les bits de poids faibles des quantités de l'image, ceci n'a guère d'impact sur la couleur et la différence est invisible à l'oeil nu. On va donc coder le message en utilisant les bits de poids faibles.
Dans la suite, on supposera que la longueur lgr des messages n'excède pas 255 caractères. Autrement dit, la longueur (écrite en binaire) peut être codée sur 8 bits. Les bits de poids faible des 8 premiers pixels de la première ligne de l'image (indexée par 0) sont modifiés pour transcrire cette longueur.
Chaque caractère du message va également être codé sur 8 bits. Les bits de poids faible des 8 premiers pixels des lignes indexées de 1 à lgr sont aussi modifiés pour transcrire chacun des lgr caractères du message.

Dans la partie 2, on précisera la méthode choisie pour dissimuler effectivement les caractères du message dans les pixels. Commençons d'abord par quelques exercices préliminaires.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from PIL import Image
```

Pour passer d'un caractère à une représentation binaire sur 8 bits, et réciproquement les fonctions suivantes seront utiles:

- `ord(c)` renvoie la valeur entière représentant le caractère `c`

- `chr(val)` est la fonction inverse de `ord(c)`
- `bin(nombre)` retourne une représentation binaire de `nombre` préfixée par `0b`
- à l'inverse, `int(chaineBin,2)` retourne la valeur entière de la représentation binaire `chaineBin`
- `zfill` , la méthode `chainenum.zfill(largeur)` retourne la chaîne numérique `chainenum` en ajoutant des `0` à gauche de sorte qu'elle soit de la largeur spécifiée. Par la suite, les représentations binaires devront être toutes codées sur 8 bits.

Question 1. Donner des exemples qui utilisent chacune de ces cinq fonctions.

```
In [2]: # Tests des cinq fonctions
```

Question 2. Écrivez une fonction **carac2bin** qui prend en entrée un caractère **c** et qui renvoie la représentation binaire codée sur **8** bits de la valeur représentant **c** .

```
In [4]: # Test carac2bin
        for car in 'hop':
            print(carac2bin(car))
```

```
01101000
01101111
01110000
```

Question 3. Écrire une fonction **bin2carac** qui réalise l'opération inverse de la fonction **carac2bin** .

```
In [6]: # Test bin2carac
        print(bin2carac('01101111'))
        print(''.join([bin2carac(cB) for cB in ('01101000','01101111','01110000')]))
```

```
o
hop
```

On rappelle le code permettant d'ouvrir une image à l'aide du module PIL et de la stocker dans un tableau numpy :

```
In [7]: img = Image.open('ricciotti_alt.png')
        im_ref = np.asarray(img, dtype=np.uint8)
```

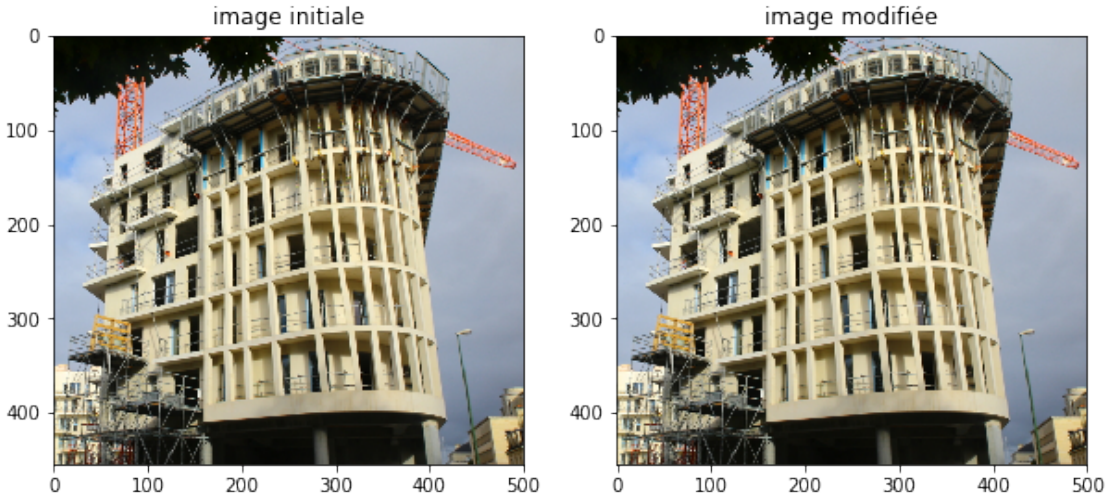
Question 4. Construire une nouvelle image **im_modif** identique à **im_ref** excepté pour les valeurs impaires des quantités de bleu dans **im_ref** qui sont décrémentées de 1.

```
In [9]: # Test
        print("Les quantités de bleu des pixels de l'image modifiée sont elles tous paires ? ",n
```

```
Les quantités de bleu des pixels de l'image modifiée sont elles tous paires ?  True
```

Question 5. Afficher l'image initiale **im_ref** et l'image modifiée **im_modif** .

In [10]:



Partie 2 - Retrouver le message caché dans une image

Le message est caché selon le procédé suivant.
Pour cacher un bit dans un pixel, on modifie la composante bleue de sorte que les composantes verte et bleue aient même parité si le bit à cacher est 0, et soient de parité différente si le bit à cacher est 1. Concrètement :

- Si le bit à cacher est 0 et que les composantes verte et bleue ont même parité, rien n'est modifié.
- Si le bit à cacher est 0 et que les composantes verte et bleue n'ont pas même parité, alors la composante bleue est incrémentée de 1 si elle est paire, et est décrémentée de 1 si elle est impaire.
- Si le bit à cacher est 1 et que les composantes verte et bleue n'ont pas même parité, rien n'est modifié.
- Si le bit à cacher est 1 et que les composantes verte et bleue ont même parité, alors la composante bleue est incrémentée de 1 si elle est paire, et est décrémentée de 1 si elle est impaire.

La longueur du message codée en binaire sur 8 bits est dissimulée dans les 8 premiers pixels de la ligne d'index 0. Le i-ème bit étant dissimulé dans le i-ème pixel comme décrit précédemment.
Le t-ème caractère codé en binaire sur 8 bits est dissimulé de même dans les 8 premiers pixels de la ligne d'index t.

```
In [11]: img = np.asarray(Image.open('ricciotti_alt.png'), dtype=np.uint8)
```

Pour commencer, cherchons la longueur du message qui est cachée dans les 8 premiers pixels de la première ligne (d'index 0) de l'image **img** .
Question 6. Récupérez les 8 premiers pixels de la première ligne dans un tableau **prem**

```
In [12]:
```

```
Out[12]: array([[ 3,  6,  1],
                [ 5,  8,  1],
                [ 4,  7,  1],
                [ 5,  8,  0],
                [ 4,  7,  0],
                [ 5,  8,  0],
                [ 6,  8,  2],
                [ 7, 10,  3]], dtype=uint8)
```

Question 7. Déterminer si les quantités vertes et bleues des 8 pixels de **prem** sont de parité différente.

```
In [13]:
```

```
Out[13]: array([ True,  True, False, False,  True, False, False,  True], dtype=bool)
```

Question 8. En déduire l'écriture en binaire de la longueur du message et sa valeur décimale.

```
In [14]:
```

```
La longueur du message a pour écriture binaire 11001001 et pour valeur 201
```

La démarche est identique pour retrouver les valeurs des caractères du message dissimulés dans les lignes suivantes de l'image. Introduisons donc une fonction auxiliaire qui rassemble les étapes précédentes.
Question 9. Écrire une fonction **decodeLigne** qui prend en entrée une ligne de pixels **ligne** est retourne la valeur décimale cachée.

```
In [16]: # Test decodeLigne
         for i in range(5):
             print(decodeLigne(img[i]))
```

```
201
73
109
112
111
```

Question 10. Écrivez une fonction **decode** qui prend en entrée une **image** contenant un message dissimulé dans les composantes bleu et verte et qui renvoie ce message. Testez votre fonction.

Partie 3 - Cacher un message dans une image

Il s'agit maintenant de réaliser l'opération inverse de la précédente.
Question 11. Écrivez une fonction **versBinaire** qui prend en entrée un message **msg** et qui retourne une liste de taille **len(msg)** dont le premier élément est la représentation en binaire de **len(msg)** codée sur 8 bits, et chaque élément suivant est la représentation en binaire codée sur 8 bits de chaque caractère du message.

```
In [19]: # Test
         versBinaire('stégano')

Out[19]: ['00000111',
          '01110011',
          '01110100',
          '11101001',
          '01100111',
          '01100001',
          '01101110',
          '01101111']
```

Question 12. Modifier la fonction précédente de sorte à ce qu'elle retourne un tableau à 2 dimensions de 0 et 1, où chaque chaîne binaire est décomposée en une ligne de 8 bits.

```
In [21]: # Test
         versBinaireV2('stégano')

Out[21]: array([[0, 0, 0, 0, 0, 1, 1, 1],
                [0, 1, 1, 1, 0, 0, 1, 1],
                [0, 1, 1, 1, 0, 1, 0, 0],
                [1, 1, 1, 0, 1, 0, 0, 1],
                [0, 1, 1, 0, 0, 1, 1, 1],
                [0, 1, 1, 0, 0, 0, 0, 1],
                [0, 1, 1, 0, 1, 1, 1, 0],
                [0, 1, 1, 0, 1, 1, 1, 1]], dtype=uint8)
```

Question 13. Écrivez une fonction **dissimule(msg,image)** qui prend en entrée un message **msg** et une image **image** et qui renvoie une image où le message est dissimulé dans les composantes bleu et verte.
Question 14. Choisissez un message (d'au plus 255 caractères) et une image d'une hauteur suffisante. Affichez l'image initiale et celle obtenue par la fonction **dissimule** .

```
In [ ]:
```

Question 15. Vérifier que votre fonction est correcte en retrouvant le message initial à l'aide de la fonction **decode** définie dans la partie 2.