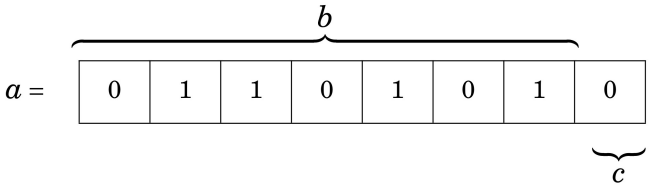
DÉFI 2 - Défi LSB - Stéganographie

Objectif : Ce défi consiste à trouver du texte dissimulé dans une image.

La problématique repose sur la méthode LSB (Least Significant Beat) dite aussi méthode des bits de poids faible vu au défi précédent.

Les images traitées sont au format `.tif` où les teintes sont codées sur un octet ($[\![0,255]\!]$). L'information est codée sur le bit de poids faible de la teinte verte de chaque pixel. Exemple pour $a = 106$:



Les fonctions à utiliser pour la lecture et la modification des images, lors de ce défi sont dans le module `imageio` :

```
import numpy as np
import imageio as im
```

Notation	Description
`im.imread( )`	Transforme une image en un tableau de 4 octets
`im.imsave( )`	Transforme un tableau de nombres en image

Les caractères sont convertible en code ASCII qui est aussi un octet. Les fonctions permettant de passer du caractère au code ASCII, puis à l'écriture binaire des nombres, sont :

Notation	Description
`ord(c)`	Donner le code ASCII d'un caractère `c` donné
`chr(n)`	Donne le caractère associé à l'octet `n` donné
`bin(n)`	Donne l'écriture binaire du nombre `n`
`int(b,2)`	retourne l'entier associé à la chaîne de caractère `b` de son écriture binaire

Exemple :

```
>>> ord('A')
65
>>> chr(65)
'A'
>>> bin(65)
'0b1000001'
>>> int('1000001',2)
65
```

Ainsi, l'image leurre est définie en modifiant chaque teinte de chaque pixel par :

- les sept bits de poids fort sont ceux de l'image leurre : `b`
- le bit de poids faible de la teinte verte est un bit du message texte.

Pour valider la réponse, il faudra aussi envoyer un script PYTHON annoté et propre à l'adresse mail au bas de la page.

Amusez-vous bien !

Vous pouvez aborder ce défi comme il vous convient ou vous laisser guider par les exercices suivants :

Exercice 1 Soit $a \in [\![0,255]\!]$ l'entier associé à un octet et les entiers `b` et `c` respectivement associés aux bits de poids fort et au bits de poids faible de `a`.

1. Justifier que

```
b=a//2
```

2. Exprimer `c` en fonction de `a`.

I. DÉFI 2 : DU TEXTE DANS UNE IMAGE !

Exercice 2 Écrire une fonction `message(d)` où `d` est une chaîne de caractères constituée de 0 et de 1 et qui :

- enlève les zéros qui sont en bout de la chaîne
- rajoute le nombre minimal de zéros afin que la longueur de `d` soit un multiple de 8
- découpe `d` par blocs de 8 bits et retourne les caractères associés.

Exercice 3 Écrire une fonction `trouver_texte(imag)` qui reconstitue le texte contenu sur le bit de poids faible de chaque teinte verte de l'image `imag`.

Remarques culturelles

Dans le cas d'une image pauvre (typiquement *uniforme*) il est plus prudent de diminuer le nombre de bits modifiés voire se limiter au bit de poids faible.

La stéganographie existe sur d'autres formats de données : fichier vidéo, fichier sonores, fichier texte ...

La principale application de la stéganographie n'est pas dans *la transmission de secret* mais dans le **Watermarking** : cela consiste à cacher un copyright au sein de l'oeuvre protégée. En cas de litige de droits, il est possible de la révéler afin de prouver son origine.

Cependant, la stéganographie est sensible à toute manipulation/retouche du fichier. Par exemple, le changement de format d'image, la compression et décompression d'un fichier, peuvent corrompre l'information cachée voire la détruire.

La détection de présence de stéganographie porte un nom anglais : la steganalysis.

La principale difficulté pour détecter la stéganographie, c'est de comprendre la façon dont l'information est cachée. C'est comme décrypter un message quand on ne connait pas l'algorithme de chiffrement qui a été utilisé ! En pratique, on effectue une recherche à partir de tous les protocoles déjà connus.

Dans certains cas, une analyse mathématique et/ou statistique peut venir compléter la détection.