

Projet : images et traitements d'images

Sommaire :

1	Organisation du travail	2
1.1	Liste des membres :	3
1.2	Étape 1 : mise en place des moyens	3
1.3	Répartition des tâches sur la vue générale	3
1.4	Pour le rendu de projet	3
2	Etude documentaire	4
2.1	Quelques formats d'image numériques	4
2.2	La couleur, synthèse additive et soustractive	4
2.3	Quelques aides pour calculer les couleurs	4
2.4	Le format de fichier BMP	5
2.5	Le codage big ou little endian	6
3	Les fichiers en python	8
3.1	Les différents types de fichier	8
3.2	Un éditeur de fichier hexadécimal	8
3.3	L'accès aux fichiers en binaire, le type bytes de python 3.x, décodage encodage des valeurs	9
a)	Prenons un premier exemple avec un fichier texte encodé en UTF8 :	9
b)	Le décodage de valeurs numériques	11
c)	Le codage des valeurs numériques	14
d)	Pour travailler en mémoire sur une image complète	14
e)	Pour approfondir l'utilisation du type bytes	14
3.4	Réaliser la recopie d'une image	15
4	Premiers travaux sur les images de tests	16
4.1	Un fichier mire pour initier l'étude	16
4.2	Analyse de la signature du fichier	17
4.3	Réalisation de fonctions élémentaires d'accès aux pixels	18
5	Travaux sur les images réelles	19
5.1	Quelques images à travailler	19
5.2	Les scripts à réaliser	20
a)	Inversion des couleurs	20
b)	Passage en niveau de gris	20
c)	Suppression de composantes couleurs	20
d)	Autres idées d'essais pour les plus rapides	20
5.3	Autres représentation des couleurs	21
6	Pour les experts : Cyber Forensic	22
7	Compléments utiles	23
7.1	Les modes d'ouverture d'un fichier	23
7.2	Attention un module peut en cacher un autre	24

1 Organisation du travail

Le projet est réalisé en équipes. Pour mener à bien le travail dans les délais impartis il sera nécessaire de se répartir le travail. L'équipe est solidaire chacun gère sa tâche mais tous avancent ensemble. L'évaluation sera globale elle portera sur les résultats obtenus et sur la qualité du travail en équipe évaluée par le professeur.

Pensez à bien sauvegarder vos travaux et à rendre les résultats dans les délais impartis c'est un excellent entraînement pour vos études supérieures.

Le présent document est un guide pour le travail à accomplir. A vous de chercher les réponses sur différentes sources : les cours assurés en classe, les ressources documentaires disponibles, les liens vers les ressources en ligne
.....

Soyez créatif !

Bon courage.

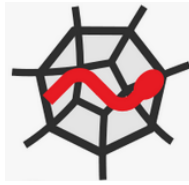
P.G

Nous on a terminés !!!

Ouais mais c'est moi qui porte tout !

Un travail d'équipe !!





Organisation de l'équipe :

1.1 Liste des membres :

-
-
-
-

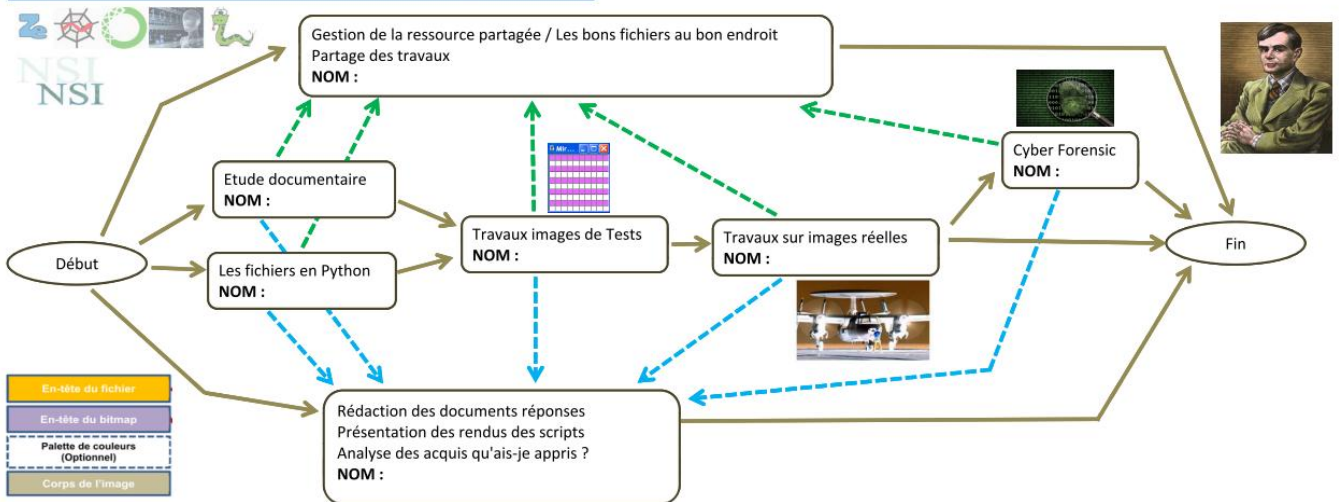
1.2 Étape 1 : mise en place des moyens

Organiser une zone de travail partagée de type Google drive ou autre accessible depuis le lycée ou bien au domicile de chacun. Cet espace sera utilisé pour stocker les documents, les résultats de travaux, les scripts en cours de réalisation.

Bien gérer cet espace et assurer le bon suivi des différents documents et leurs différentes versions est une tâche très importante.

1.3 Répartition des tâches sur la vue générale

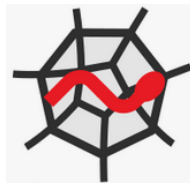
Projet : images et traitements d'images



1.4 Pour le rendu de projet

NSI_RENDUS_PROJETS.pdf





2 Etude documentaire

Ce projet porte sur l'analyse et le traitement d'images numériques. Les scripts seront réalisés en python. Avant de commencer nous allons étudier quelques aspects du domaine des images numériques.



Conseil : Cette partie peut être traitée indépendamment.

Les réponses aux questions seront données dans un fichier Word ou équivalent.

2.1 Quelques formats d'image numériques



Q1. Indiquez quelques formats d'images utilisés dans le monde numérique.

Q2. Précisez la différence entre une image vectorielle et une image bitmap.

2.2 La couleur, synthèse additive et soustractive



Q3. Documentez-vous sur la colorimétrie et précisez ce que signifie synthèse additive et synthèse soustractive.

Q4. Indiquez un périphérique dans chacune des synthèses.

Même en couleurs inversées on nous reconnait !

2.3 Quelques aides pour calculer les couleurs

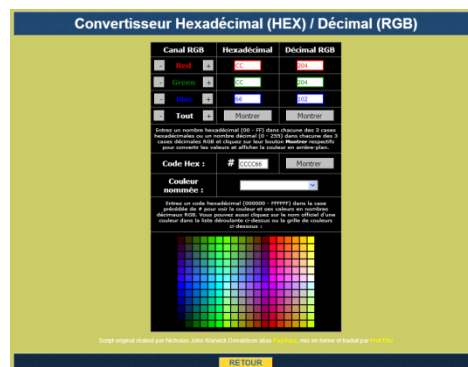
Pour calculer les couleurs le lien vers un logiciel à installer chez soi :

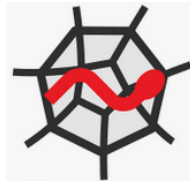
La "Boîte à couleurs".

<http://www.colovid.be/ColorBox.htm>

Ou bien un site à consulter en ligne :

<http://www.proftnj.com/RGB3.htm>





2.4 Le format de fichier BMP

Une fiche résumant les caractéristiques des fichiers bmp est consultable en ligne ici :

[www.apprendre-en-ligne.net > info > images > formatbmp](http://www.apprendre-en-ligne.net/info/images/formatbmp)

Voir également la copie du fichier [formatbmp.pdf](#) dans les ressources.



Le format BMP est un format d'image bitmap développé conjointement par Microsoft et IBM. Les pixels sont stockés sous forme de tableau de points. On gère les couleurs soit en couleur vraie soit grâce à une palette indexée.

En-tête du fichier

L'entête du fichier fournit des informations sur le type de fichier (Bitmap), sa taille et indique où commencent les informations concernant l'image à proprement parler.

En-tête du bitmap

L'entête de l'image fournit des informations sur l'image, notamment ses dimensions et ses couleurs.

La palette de couleur (éventuellement)

Si le nombre de bits utilisés pour coder la couleur de chaque pixel est inférieur ou égal à 8 (soit 256 couleurs), le deuxième en-tête est suivi d'une **table des couleurs** utilisées dans l'image.

Chaque couleur de la palette est codée sur 4 octets :

- 1 octet pour l'intensité du bleu (00 à FF)
- 1 octet pour l'intensité du vert (00 à FF)
- 1 octet pour l'intensité du rouge (00 à FF)
- 1 octet réservé (XX)

Le corps de l'image

Le codage de l'image se fait en écrivant successivement les bits correspondant à chaque pixel, ligne par ligne en commençant par le pixel en bas à gauche.

En-tête du fichier (BITMAPFILEHEADER)				
Adresse	Taille (octets)	Nom	Valeur standard (hex)	Signification
00000000	2	bfType	42 4D	Signature du fichier. BM=Bitmap windows, BA= Bitmap OS2, CI=icône couleur OS2, CP=pointeur de couleur OS2, IC= icône OS2, PT=pointeur OS2
00000002	4	bfSize	00 00	Taille totale du fichier en octets
00000006	2	bfReserved1	00 00	Champ réservé
00000008	2	bfReserved2	00 00	Champ réservé
0000000A	4	bfOffBits	36 00 00 00	Adresse de la zone de définition de l'image

En-tête du bitmap (BITMAPINFOHEADER)				
Adresse	Taille (octets)	Nom	Valeur standard (hex)	Signification
0000000E	4	biSize	28 00 00 00	Taille en octets de cet en-tête BITMAPINFOHEADER
00000012	4	biWidth	00 00 00 00	Largeur de l'image en pixels
00000016	4	biHeight	00 00 00 00	Hauteur de l'image en pixels
0000001A	2	biPlanes	01 00	Nombre de plans
0000001C	2	biBitCount	00 00	Nombre de bits par pixel
0000001E	4	biCompression	00 00 00 00	Type de compression : 0=pas de compression, 1=compressé à 8 bits par pixel, 2=4bits par pixel.
00000022	4	biSizeImage	00 00 00 00	Taille en octets des données de l'image
00000026	4	biXpelsPerMeter	00 00 00 00	Résolution horizontale en pixels par mètre
0000002A	4	biYpelsPerMeter	00 00 00 00	Résolution verticale en pixels par mètre
0000002E	4	biClrUsed	00 00 00 00	Nombre de couleurs dans l'image : 0=maximum possible. Si une palette est utilisée, ce nombre indique le nombre de couleurs de la palette
00000032	4	biClrImportant	00 00 00 00	Nombre de couleurs importantes. 0= toutes importantes

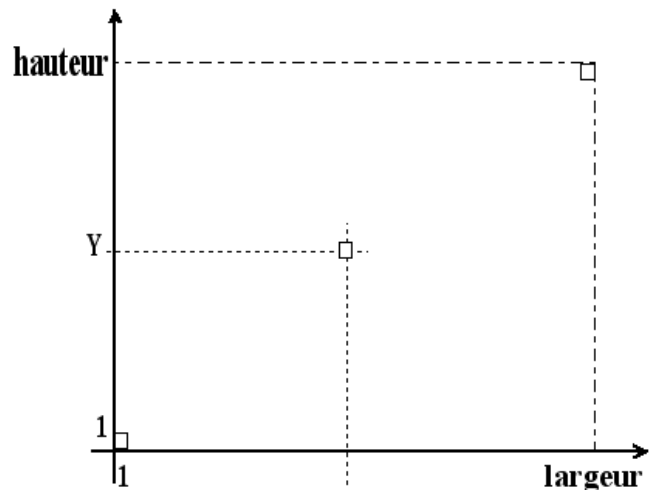
Nom : papillon24.bmp
 Taille du fichier : 49208 octets
 Dimensions : 128 x 128 (pixels)
 Nombre de pixels : 16384
 Couleurs : 24 bits (16777216 couleurs)

En-tête de fichier

 En-tête de bitmap

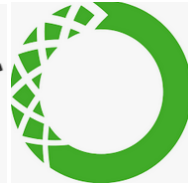
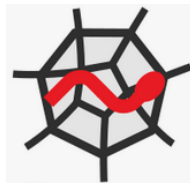
 Corps de l'image

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	42	4D	38	C0	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	00	00	00	00	00	00	00	00	01	00	18	00	00	00
00000020	00	00	02	C0	00	00	C3	0E	00	00	C3	0E	00	00	00	00
00000030	00	00	00	00	00	00	37	83	5F	38	82	60	38	82	5D	38
00000040	82	5F	37	82	5E	37	70	7A	3A	80	5D	33	80	5C	33	7F
00000050	5C	31	7F	59	31	7E	59	2F	7D	59	2D	78	58	2A	79	59
00000060	27	78	54	27	77	54	26	76	53	25	75	52	74	51	73	51
00000070	78	56	2E	7A	54	2E	7A	53	2A	79	53	30	7D	5D	33	7D
00000080	5F	35	7E	61	36	7F	63	36	7F	67	36	81	68	37	82	6D
00000090	3A	83	6D	3B	84	70	3C	87	70	40	87	71	42	89	72	44
000000A0	8A	77	46	8C	78	49	8D	79	49	8F	79	4B	90	7C	4E	91
000000B0	7E	50	93	7F	54	94	80	58	94	80	5B	95	81	5B	95	80
000000C0	5B	96	82	5B	96	7E	5D	95	7D	5D	95	7D	5E	95	7C	5B
000000D0	95	7A	5D	95	7C	5F	95	7A	5F	95	79	60	94	79	60	94
000000E0	78	60	93	78	61	91	78	62	91	76	60	8F	74	5E	8D	72
000000F0	5B	8D	71	59	8A	70	55	8A	6B	51	89	6B	4E	88	6A	49



L'image est constituée (largeur x hauteur) pixels
la référence (1,1) est en bas à droite.





Chaque pixel est codé avec trois octets R,V,B. Attention dans le fichier chaque ligne doit être un multiple de 4 elle est donc éventuellement complétée par des octets à 0 : 0x00.

2.5 Le codage big ou little endian

Pour les nombres codés sur plusieurs octets le problème de l'ordre dans lequel on récupère ces octets dans la mémoire de l'ordinateur est appelé l'endianness.

C'est l'ordre des octets pour les entiers de 16 32 ou 64 bits. C'est l'adresse de l'octet de poids FAIBLE pour le **LITTLE ENDIAN**, l'octet suivant est l'octet de poids fort. C'est l'adresse de l'octet de poids FORT pour le **BIG ENDIAN**.

Big Endian

12	34	56	78
octet	octet	octet	octet

Little Endian

78	56	34	12
octet	octet	octet	octet

chortle.ccsu.edu

[Little endian](#)
www.lbgi.fr > ~ripp > help > unix > endian

Voir ici : <https://developer.mozilla.org/fr/docs/Glossaire/Endianness> dont est issu cet exemple :

Exemples avec le nombre `0x12345678` (i.e. 305 419 896 en décimal) :

- *little-endian* : `0x78 0x56 0x34 0x12`
- *big-endian* : `0x12 0x34 0x56 0x78`



Q5. Sur combien d'octets sont codés dans les fichiers BMP les champs :
 bfOffBits i.e. : adresse de la zone de définition de l'image
 biWidth i.e. : Largeur de l'image en pixels
 biHeight i.e. : Largeur de l'image en pixels

Q6. Ces nombres sont-ils codés en little endian ou big endian ?

Nous utiliserons des images sans palettes de couleurs. Les pixels de l'image sont donc codés individuellement en RVB sur trois octets à la suite les uns des autres. Ils sont codés ligne par ligne en commençant en bas à gauche et en finissant en haut à droite. La contrainte du format impose que le codage de chaque ligne de pixels contienne un nombre d'octets multiple de 4. Les lignes sont complétées par des octets 0x00 si besoin.

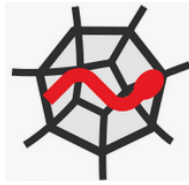


Q7. A partir de cette description déterminer la relation permettant de trouver la position du premier octet du triplet RVB dans la zone image (en réalité dans l'ordre inverse BVR dans le fichier).

Conseil : Cette partie plus délicate mérite l'attention de toute l'équipe. **Pour cette question voir compléments ici :**

[NSI_PROJETS_TRAITEMENTS_IMAGES_COMPLEMENTS.pdf](#)





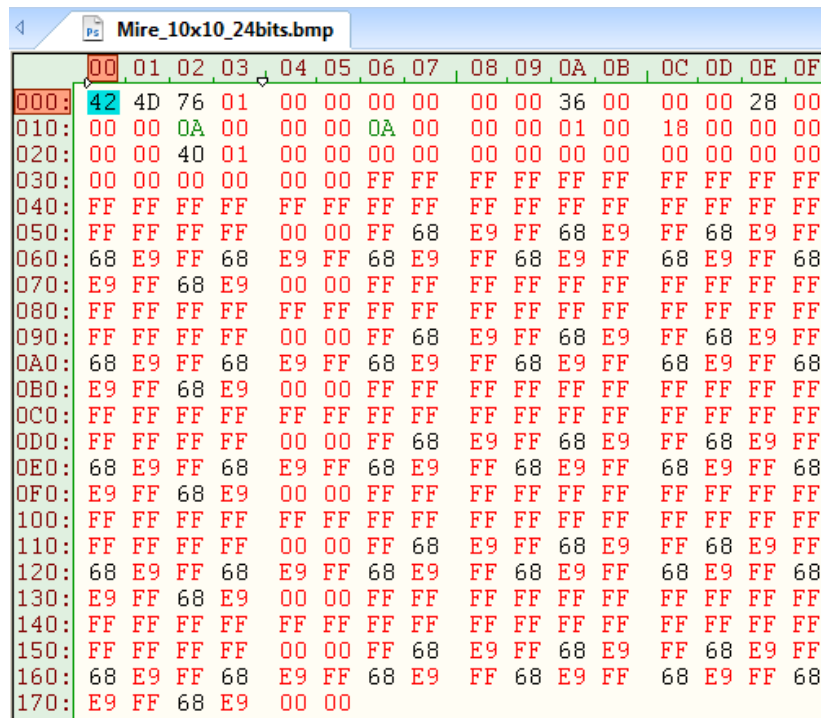
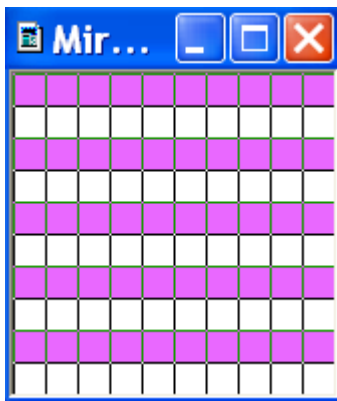
Cette position ou offset_octet_RVB doit être établie en fonction de X, Y et de la largeur de l'image notée L.
 $\text{offset_octet_RVB} = f(X, Y, L)$

Pour vérifier votre fonction voilà une liste de résultats à tester :

X	Y	L	offset_octet_RVB
1	1	10	#00 0
10	1	10	#1B 27
1	2	10	#20 32
8	5	9	#185 133
6	4	11	#7B 123
18	14	24	#03DB 987
100	100	512	#25329 152361

Version corrigée 14/11/19

Une image pour réfléchir à votre algorithme

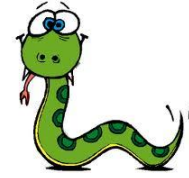
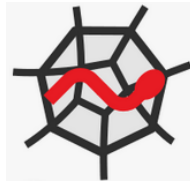


Q8. Donner la relation finale calculant l'offset du premier octet du triplet RVB d'un pixel (X,Y) dans le fichier complet dénommé **offset_fichier_RVB**



Script1. Programmer la fonction python réalisant le calcul de la valeur offset_fichier_RVB à partir de X, Y, L, bfOffBits.





3 Les fichiers en python

3.1 Les différents types de fichier

Conseil : Cette partie peut être traitée indépendamment.

Les images que nous allons travailler sont stockées dans des fichiers. Nous étudions ici les différents types de fichiers. Puis comment python permet d'accéder à ces fichiers et plus particulièrement aux fichiers binaires contenant nos images.

[Lire une introduction](#) sur les fichiers et python puis répondre aux questions ci-dessous.



Q9. Un simple examen du contenu binaire permet-il d'en déduire la nature des informations contenus dans un fichier ?



Q10. Quelles sont les ressemblances et les différences entre un simple fichier texte écrit sous Notepad par exemple et des fichiers json ou CSV ?

Q11. Que signifie CSV ?

3.2 Un éditeur de fichier hexadécimal

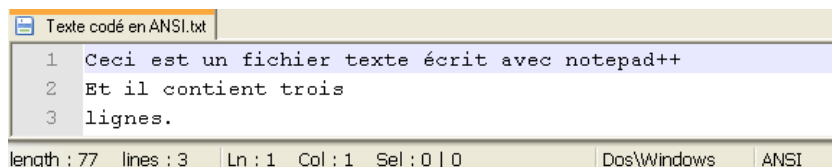
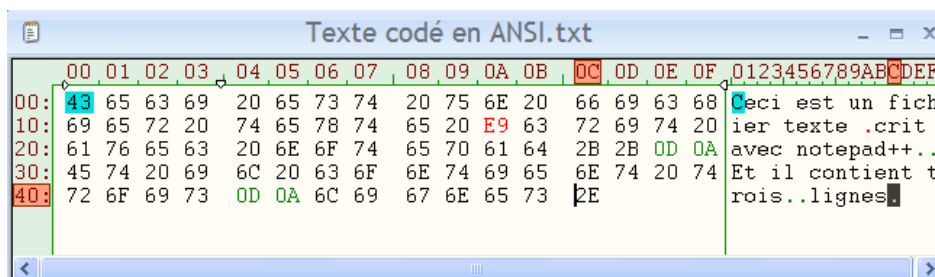
Un éditeur de fichier hexadécimal permet une visualisation du binaire d'un fichier quelque soit son type. Il en existe plusieurs, nous utiliserons le freeware hexedit :

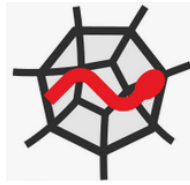
<https://www.techworld.com/download/developer-programming/hexedit-630-3249772/>



Q12. Ouvrir avec hexedit le fichier Texte codé en ANSI.txt que peut-on dire des fins de lignes comment sont-elles codées ?

Q13. Comparer les fichiers Texte codé en ANSI.txt et Texte codé en UTF8.txt Expliquer les différences.





3.3 L'accès aux fichiers en binaire, le type bytes de python 3.x, décodage encodage des valeurs

Les fichiers sont au final une collection d'octets. Pour décoder cette collection il faut connaître l'encodage et donc la nature du fichier.

a) Prenons un premier exemple avec un fichier texte encodé en UTF8 :

Prenons comme exemple un texte contenant volontairement beaucoup de caractères accentués :

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
00:	41	75	20	6D	65	6E	75	20	64	75	20	63	C3	A9	6C	C3	Au menu du c..l.
10:	A9	72	69	20	61	76	65	63	20	64	75	20	67	72	75	79	.ri avec du gruy
20:	C3	A8	72	65	20	6D	61	6E	67	C3	A9	20	61	75	20	62	..re mang.. au b
30:	6F	72	64	20	64	65	20	6C	61	20	72	69	76	69	C3	A8	ord de la rivi..
40:	72	65	20	65	74	20	C3	A0	20	6C	61	20	66	69	6E	20	re et .. la fin
50:	75	6E	65	20	74	61	73	73	65	20	64	65	20	63	61	66	une tasse de caf
60:	C3	A9	2E														...

Nous allons lire ce fichier texte et afficher ensuite le résultat lu brut, puis avec le bon décodage.

Voilà le script python utilisé :

test_lecture_fichier_int.py

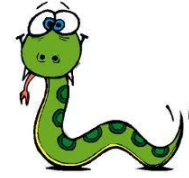
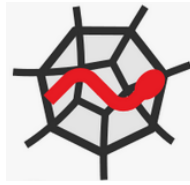
```

1  #-*- coding: utf-8 -*-
2  #
3  # test_lecture_fichier_int.py
4  #
5  # P.G Novembre 2019
6
7  # Adapter le dossier de travail si besoin
8  from os import chdir
9  path = r'P:\PRO\USB\NSI\NSI STEGANOGRAPHIE'
10 chdir(path)
11
12 # Chargement des bibliothèques
13 import struct
14 from os import stat
15
16 # Récupération de la taille du fichier cible
17 nom_fichier = "Texte avec accents codé en UTF8.txt"
18 info_fichier = stat(nom_fichier)
19 taille_totale_du_fichier = info_fichier.st_size

```

Ne pas oublier dans la suite !!





```
20
21 # Ouverture du fichier
22 fich=open(nom_fichier,"rb")
23 # Positionnement de la tête de lecture en position 0
24 fich.seek(0)
25 # Lecture de tous les bytes du fichier
26 donnees_fichier = fich.read(taille_totale_du_fichier)
27 # Fermeture du fichier
28 fich.close()
29
30 # Affichage et comparaison des résultats
31 print("\nFichier : [texte avec accents codé en UTF8.txt]\n")
32 print("Lecture de {:d} octets affichage au format bytes : \n"
33       | .format(taille_totale_du_fichier))
34 print(donnees_fichier,"\n")
35 print("Après décodage en UTF8 : \n")
36 print(donnees_fichier.decode('utf8'))
```

Les résultats :

```
>>> (executing lines 1 to 36 of "test_lecture_fichier_int.py")
```

```
Fichier : [texte avec accents codé en UTF8.txt]
```

```
Lecture de 99 octets affichage au format bytes :
```

```
b'Au menu du c\xc3\xa9léri avec du gruy\xca8re mang\xca9\xca9 au bord de la rivi\xca8re et \xc3\xa0 la fin une tasse de c af\xca9.'
```

Après décodage en UTF8 :

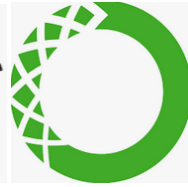
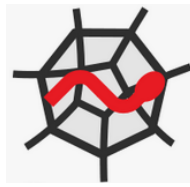
```
Au menu du céleri avec du gruyère mangé au bord de la rivière et à la fin une tasse de café.
```

Tout rentre dans l'ordre en affichant le contenu avec le bon format de décodage !!



Le paragraphe suivant explorera le décodage des valeurs numériques.





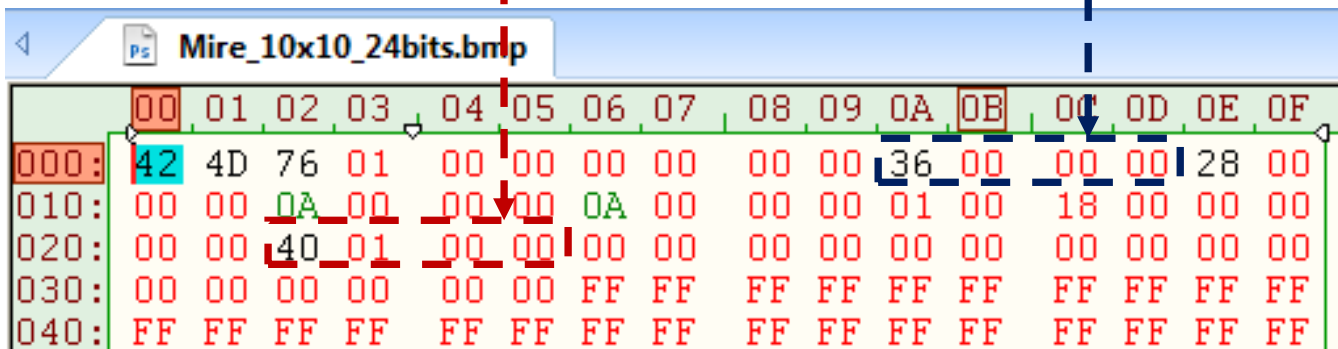
b) Le décodage de valeurs numériques

Pour illustrer la méthode de lecture nous allons lire le champ contenant la taille de l'image dans l'en tête du fichier .bmp

En examinant l'en-tête du fichier nous repérons deux champs qui vont nous être utiles, nous allons lire la taille en octets des données de l'image à l'adresse 0x22 du fichier.

0000000A	4	bfOffBits	36 00 00 00	Adresse de la zone de définition de l'image
00000022	4	biSizeImage	00 00 00 00	Taille en octets des données de l'image

Une vue hexadécimale du fichier nous indique le contenu de ces deux champs :



Script python utilisé

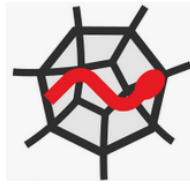
test_lecture_fichier.py

```

1 #
2 #
3 # test_lecture_fichier.py
4 #
5 # P.G Novembre 2019
6
7 # Adapter le dossier de travail si besoin
8 from os import chdir
9 path = r'P:\PRO\USB\NSI\NSI STEGANOGRAPHIE'
10 chdir(path)
11
12 # Chargement des bibliothèques
13 import struct
14
15 # Ouverture du fichier
16 fich=open('Mire_10x10_24bits.bmp',"rb")
17 # Positionnement de la tête de lecture en position 0x22
18 fich.seek(0x22)
19 # Lecture des quatre bytes encodant la taille de l'image en octets
20 donnees_fichier = fich.read(4)
21 # Fermeture du fichier
22 fich.close()

```





```
23
24 # Décodage de l'entier positif sur 4 bytes, (type unsigned int du langage c)
25 # codé en little endian
26 valeur_taille = struct.unpack('<I', donnees_fichier)[0]
27
28 # Affichage des résultats
29 print("Lecture de la taille de l'image Mire_10x10_24bits.bmp en octets")
30 print("Les quatre bytes lus : ",donnees_fichier)
31 print("De type : ",type(donnees_fichier))
32 print("Valeur obtenue après décodage : ",valeur_taille)
```

Résultats obtenus

```
>>> (executing lines 1 to 32 of "test_lecture_fichier.py")
Lecture de la taille de l'image Mire_10x10_24bits.bmp en octets
Les quatre bytes lus : b'@\x01\x00\x00'
De type : <class 'bytes'>
Valeur obtenue après décodage : 320
```

Interprétation

Nous voyons dans le dump du fichier que la valeur du champ qui nous intéresse est :

0x40 0x01 0x00 0x00 indiqué par la chaîne de bytes : `b'@\x01\x00\x00'`

Comment transformer ces quatre octets en nombre ?

Il faut les décoder avec la méthode `unpack` du module `struct`. Cette méthode est capable d'interpréter le flot d'octets comme un nombre codé en binaire avec différents formats du langage C. (voir la documentation officielle de python 3 :

<https://docs.python.org/3/library/struct.html>)

Le décodage est réalisé par l'instruction :

```
struct.unpack('<I', donnees_fichier)[0]
```

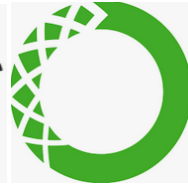
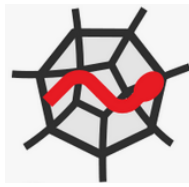
Nous remarquons que la méthode `unpack` renvoie un tuple avec un seul élément que l'on peut sélectionner directement avec `[0]` :

```
>>> donnees_fichier
b'@\x01\x00\x00'

>>> struct.unpack('<I', donnees_fichier)
(320,)

>>> struct.unpack('<I', donnees_fichier)[0]
320
```





Pour interpréter correctement les octets il faut préciser à `unpack` comment si prendre donc quel est le type de données à récupérer et le boutisme utilisé little ou big endian.

```
struct.unpack('<I', donnees_fichier)[0]
```

Voilà comment interpréter '<I' :

Format	C Type	Python type	Standard size
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer	4
f	float	float	4
d	double	float	8

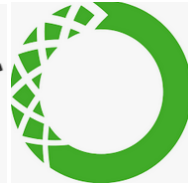
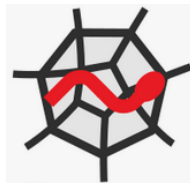
Character	Byte order	Size	Alignment
@	native	native	native
=	native	standard	none
<	little-endian	standard	none
>	big-endian	standard	none
!	network (= big-endian)	standard	none



Q14. La suite d'octets lue est donc 0x40 0x01 0x00 0x00. Vérifier que le décodage donne bien 320 en considérant un type unsigned int.

Q15. Vérifier ensuite que la taille totale du fichier correspond bien à celle indiquée par l'OS à savoir 374 bytes.





c) Le codage des valeurs numériques

On utilise la méthode pack du module struct.

Pour modifier un seul octet on peut procéder comme suit :

```
# Ouverture du fichier en modification "rb+"
fich=open('Copie de Mire_10x10_24bits.bmp','rb+')
fich.seek(0x22)
fich.write(struct.pack('B',0x44))
fich.close()
```

d) Pour travailler en mémoire sur une image complète

Pour travailler en mémoire sur une image complète et pour limiter les accès disque il peut être plus simple de recopier l'image dans une liste. Puis après modification de sauvegarder cette liste dans le nouveau fichier. Voir l'exemple ci-dessous :

```
# Ouverture du fichier cible
image_a_traiter = input("Donner le nom complet du fichier à traiter : ")
if image_a_traiter=='':
    image_a_traiter = "Mire.bmp"
fichier_origine = open(image_a_traiter,"rb")

info_fichier = stat(image_a_traiter)
taille_du_fichier = info_fichier.st_size
print("\nNom du fichier traité : ",image_a_traiter)

# Recopie dans une liste
for i in range(taille_du_fichier):
    image_totale.append(fichier_origine.read(1))

# Fermeture du fichier
fichier_origine.close()
```

Et la recopie après traitement :

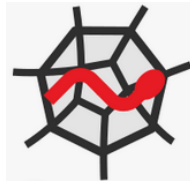
```
# A partir de l'image travaillée en local dans la liste
# on la recopie dans un fichier
fichier_resultat=open("test.bmp","wb")
for byte in image_totale:
    fichier_resultat.write(byte)
fichier_resultat.close()
```

e) Pour approfondir l'utilisation du type bytes

De nombreux sites, en plus de la documentation officielle python, permettent d'approfondir nos connaissances sur le type bytes et les mécanismes de codage ou décodage.

Citons : <http://sametmax.com/le-type-bytes-cest-pas-du-texte/> site excellent techniquement au langage parfois un peu 'imagé'.





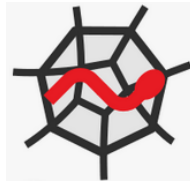
3.4 Réaliser la recopie d'une image.

Pour travailler sur une image il est préférable au préalable de la recopier pour sauvegarder l'image originale. C'est très simple à réaliser il n'y a aucun traitement on se contente d'ouvrir le fichier source en lecture, le fichier destination en écriture et de recopier les bytes :

Lecture fichier BMP et recopie.py

```
1 #-*- coding: utf-8 -*-
2 #
3 # Lecture fichier BMP et recopie.py
4
5 # Pour se positionner dans le répertoire de travail
6 from os import chdir
7 path = r'P:\PRO\USB\NSI\NSI STEGANOGRAPHIE'
8 chdir(path)
9
10 # Chargement des modules
11 from struct import pack
12 from os import stat
13
14 # Saisie du nom du fichier d'origine et ouverture en mode lecture binaire
15 image_origine = input("Entrez le nom sans l'extension de l'image d'origine : ")
16 fichier_origine = open(image_origine+'.bmp',"rb")
17
18 # Création du nom du fichier de l'image recopiée et ouverture en écriture
19 # binaire
20 image_recopiee = image_origine + '_copie' + '.bmp'
21 fichier_resultat=open(image_recopiee,"wb")
22
23 print("L'image modifiée sera dans le fichier : ",image_recopiee)
24
25 # Lecture de la taille du fichier d'entrée
26 info_fichier = stat(image_origine+'.BMP')
27 taille_du_fichier = info_fichier.st_size
28
29 print("La taille du fichier est de : ",taille_du_fichier)
30
31 # Recopie avec une boucle sur tous les octets du fichier d'entrée
32 for i in range(taille_du_fichier):
33     bytes = fichier_origine.read(1) # read the next /byte/
34     fichier_resultat.write(bytes)
35
36 # Fermeture des fichiers
37 fichier_origine.close()
38 fichier_resultat.close()
```



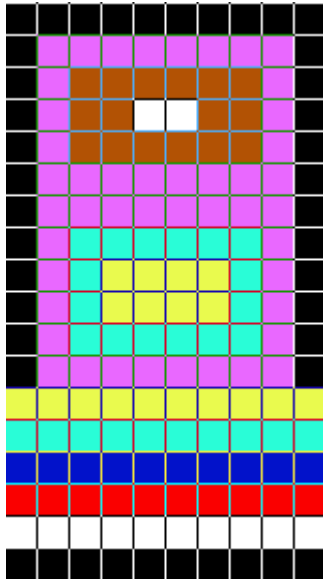


4 Premiers travaux sur les images de tests

4.1 Un fichier mire pour initier l'étude

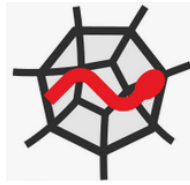
Dans cette partie il faut mettre au point les différents scripts et fonctions python nécessaires à l'accomplissement des travaux du chapitre suivant sur des images réelles.

Pour simplifier l'étude nous travaillons sur l'image mire.bmp :



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000:	42	4D	76	02	00	00	00	00	00	00	36	00	00	00	28	00
010:	00	00	0A	00	00	00	12	00	00	00	01	00	18	00	00	00
020:	00	00	40	02	00	00	00	00	00	00	00	00	00	00	00	00
030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050:	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
060:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
070:	FF	FF	FF	FF	00	00	00	00	FA	00	00	FA	00	00	FA	00
080:	00	FA	00	00	FA	00	00	FA	00	00	FA	00	00	FA	00	00
090:	FA	00	00	FA	00	00	CA	12	02	CA	12	02	CA	12	02	CA
0A0:	12	02	CA	12	02	CA	12	02	CA	12	02	CA	12	02	CA	12
0B0:	02	CA	12	02	00	00	D7	FD	29	D7	FD	29	D7	FD	29	D7
0C0:	FD	29	D7	FD	29	D7	FD	29	D7	FD	29	D7	FD	29	D7	FD
0D0:	29	D7	FD	29	00	00	4E	FA	E9	4E	FA	E9	4E	FA	E9	4E
0E0:	FA	E9	4E	FA	E9	4E	FA	E9	4E	FA	E9	4E	FA	E9	4E	FA
0F0:	E9	4E	FA	E9	00	00	00	00	00	FF	68	E9	FF	68	E9	FF
100:	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68
110:	E9	00	00	00	00	00	00	00	00	FF	68	E9	D7	FD	29	D7
120:	FD	29	D7	FD	29	D7	FD	29	D7	FD	29	D7	FD	29	FF	68
130:	E9	00	00	00	00	00	00	00	00	FF	68	E9	D7	FD	29	4E
140:	FA	E9	4E	FA	E9	4E	FA	E9	4E	FA	E9	D7	FD	29	FF	68
150:	E9	00	00	00	00	00	00	00	00	FF	68	E9	D7	FD	29	4E
160:	FA	E9	4E	FA	E9	4E	FA	E9	4E	FA	E9	D7	FD	29	FF	68
170:	E9	00	00	00	00	00	00	00	00	FF	68	E9	D7	FD	29	D7
180:	FD	29	D7	FD	29	D7	FD	29	D7	FD	29	D7	FD	29	FF	68
190:	E9	00	00	00	00	00	00	00	00	FF	68	E9	FF	68	E9	FF
1A0:	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68
1B0:	E9	00	00	00	00	00	00	00	00	FF	68	E9	FF	68	E9	FF
1C0:	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68
1D0:	E9	00	00	00	00	00	00	00	00	FF	68	E9	04	52	B2	04
1E0:	52	B2	04	52	B2	04	52	B2	04	52	B2	04	52	B2	FF	68
1F0:	E9	00	00	00	00	00	00	00	00	FF	68	E9	04	52	B2	04
200:	52	B2	FF	FF	FF	FF	FF	FF	04	52	B2	04	52	B2	FF	68
210:	E9	00	00	00	00	00	00	00	00	FF	68	E9	04	52	B2	04
220:	52	B2	04	52	B2	04	52	B2	04	52	B2	04	52	B2	FF	68
230:	E9	00	00	00	00	00	00	00	00	FF	68	E9	FF	68	E9	FF
240:	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68	E9	FF	68
250:	E9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
260:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
270:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00





4.2 Analyse de la signature du fichier

Les fichiers peuvent être identifiés avec les extensions de leurs noms comme mire.bmp mais il est très facile de changer l'extension en renommant le fichier par exemple. Une signature est intégrée dans le contenu même du fichier directement en binaire que 1 ou plusieurs octets.

Une liste de signatures peut être vue ici : https://www.garykessler.net/library/file_sigs.html

Pour le fichier bitmap BMP nous avons les informations suivantes :

42 4D

BM

BMP, DIB Windows (or device-independent) bitmap image

NOTE: Bytes 2-5 contain the file length in little-endian order.

L'analyse Cyber Forensic consiste à recueillir des indices ou preuves d'intrusions directement dans les contenus numériques : fichiers, disque durs, mémoires vive des ordinateurs.....



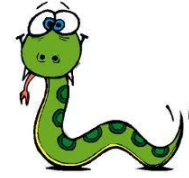
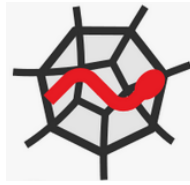
Q16. Analyse forensic du fichier mire.bmp, répondre aux questions suivantes :

- Vérifier le type du fichier
- Vérifier sa longueur en octet
- Donner l'adresse du début de l'image
- Quel est le nombre de pixels en largeur ?
- Quel est le nombre de pixels en hauteur ?



Q17. A partir du nombre de pixels par ligne donner le nombre d'octets utilisés pour coder ces pixels. Ce nombre est-il un multiple de 4 ? Si non combien d'octet faut-il ajouter à chaque ligne pour vérifier la norme de représentation.





4.3 Réalisation de fonctions élémentaires d'accès aux pixels

Conseil : Chacun de ces scripts devra être codé et vérifié indépendamment. Les résultats des essais individuels seront présentés dans le compte rendu final.



Script2. Réaliser un script python qui lit et affiche les paramètres d'une image Hauteur, Largeur, Taille totale du fichier, Adresse de la zone de définition image ...



Script3. Réaliser un script python avec une fonction qui permet de lire les valeurs RVB d'un pixel de coordonnées (X,Y)

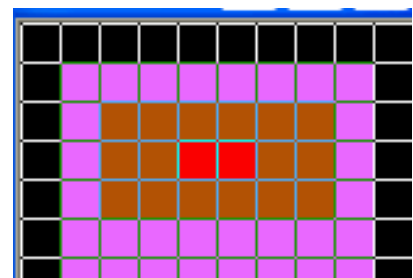


Script4. Réaliser un script python qui permet de modifier les couleurs d'un pixel de coordonnées (X,Y)



Script5. Utiliser le script précédent pour tirer dans la cible :

Facile !!!



5 Travaux sur les images réelles¹

Conseil : Pour traiter les images réelles il faudra incorporer les travaux des scripts précédents dans le script `PROJET_IMAGE_CHASSIS.py`

5.1 Quelques images à travailler

Voilà quelques images pour réaliser nos expérimentations :



[Minion.bmp](#)



[Abeille Flandre.bmp](#)



[hawkeye.bmp](#)

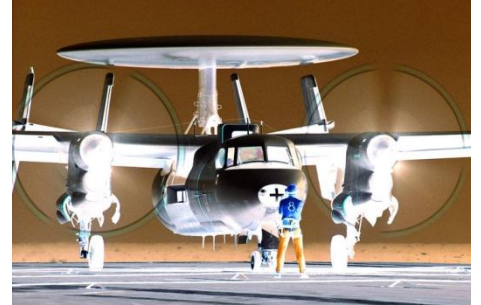


¹ Les travaux de ce paragraphe ont été très inspirés du site suivant
<http://www.tangentex.com/TraitementImages.htm#Par1>

5.2 Les scripts à réaliser

a) Inversion des couleurs

Script6. Réaliser une inversion des couleurs pour cela il faut recopier l'image en inscrivant pour chacune des valeurs de couleurs la valeur complémentaire $255 - R$, $255 - V$, $255 - B$



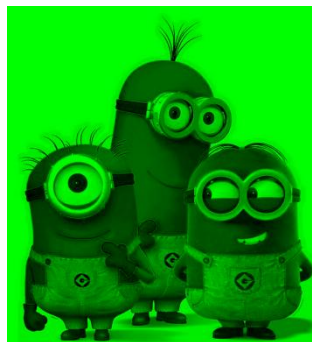
b) Passage en niveau de gris

Script7. Réaliser une transformation en niveau de gris pour cela chaque pixel R, V, B est remplacé par le triplet G, G, G où
 $G = 0.2125 \cdot R + 0.7154 \cdot G + 0.0721 \cdot B$



c) Suppression de composantes couleurs

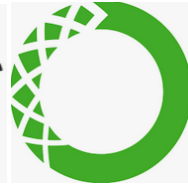
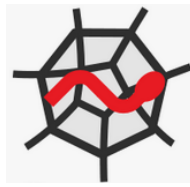
Script8. Réaliser un script qui ne garde qu'une seule des trois composantes sur une image.



d) Autres idées d'essais pour les plus rapides

D'autres modifications et opérations telles que le filtrage sont décrites sur le site mis en référence.





5.3 Autres représentation des couleurs

Il existe d'autres représentations des couleurs que les triplets de valeurs R, V, B. Les calculs de transformation d'un système de représentation dans un autre sont dévolus à la librairie python colorsys : <https://docs.python.org/3/library/coloursys.html>

Pour la théorie voir par exemple ici [representationCouleur.pdf](#)

coloursys — Conversions between color systems

Source code: [Lib/coloursys.py](#)

The `coloursys` module defines bidirectional conversions of color values between colors expressed in the RGB (Red Green Blue) color space used in computer monitors and three other coordinate systems: YIQ, HLS (Hue Lightness Saturation) and HSV (Hue Saturation Value). Coordinates in all of these color spaces are floating point values. In the YIQ space, the Y coordinate is between 0 and 1, but the I and Q coordinates can be positive or negative. In all other spaces, the coordinates are all between 0 and 1.

See also: More information about color spaces can be found at <http://poynton.ca/ColorFAQ.html> and <https://www.cambridgeincolour.com/tutorials/color-spaces.htm>.

The `coloursys` module defines the following functions:

```
coloursys.rgb_to_yiq(r, g, b)
```

Convert the color from RGB coordinates to YIQ coordinates.

```
coloursys.yiq_to_rgb(y, i, q)
```

Convert the color from YIQ coordinates to RGB coordinates.

```
coloursys.rgb_to_hls(r, g, b)
```

Convert the color from RGB coordinates to HLS coordinates.

```
coloursys.hls_to_rgb(h, l, s)
```

Convert the color from HLS coordinates to RGB coordinates.

```
coloursys.rgb_to_hsv(r, g, b)
```

Convert the color from RGB coordinates to HSV coordinates.

```
coloursys.hsv_to_rgb(h, s, v)
```

Convert the color from HSV coordinates to RGB coordinates.



6 Pour les experts : Cyber Forensic

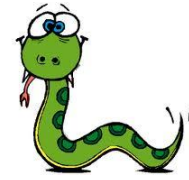
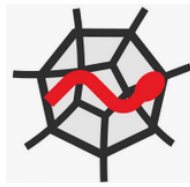
Une connaissance approfondie permet d'exploiter certaines caractéristiques des contenus informatiques à d'autres fins que celles initialement prévues.



Dans notre cas le travail consiste à étudier la faisabilité d'exploiter les valeurs 0x00 imposée par le codage des fichiers de type bmp quand il n'y a pas un nombre d'octets multiple de quatre sur chaque ligne. La question auquel vous devez répondre est la suivante :

EXPERT1. Est-il possible de ce servir de ces octets 'inutiles' pour transporter une charge utile ou payload (dans le jargon informatique). Cette charge utile étant un message encodé en ASCII par exemple, éventuellement lui-même chiffré. Et ce sans altérer les capacités d'afficher l'image modifiée ?





7 Compléments utiles

7.1 Les modes d'ouverture d'un fichier

Quelques exemples d'ouverture de fichiers :

```
fichier_origine = open(image_origine+'.bmp', "rb")
```

```
fichier_resultat=open(image_recopiee, "wb")
```

"rb" indique une ouverture en lecture et mode binaire.

"wb" indique une ouverture en mode écriture et binaire.

Nous pouvons invoquer l'aide de la fonction open :

```
>>> help(open)
```

```
Help on built-in function open in module io:
```

```
open(...)
```

```
open(file, mode='r', buffering=-1, encoding=None,  
      | errors=None, newline=None, closefd=True, opener=None) -> file object
```

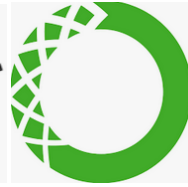
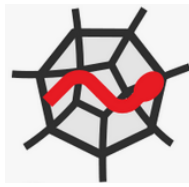
```
Open file and return a stream. Raise IOError upon failure.
```

```
=====
```

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	create a new file and open it for writing
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newline mode (deprecated)

```
=====
```





7.2 Attention un module peut en cacher un autre

Attention cependant il y a dans la bibliothèque os (Operating System) une autre fonction open qui ne possède pas les mêmes arguments d'appel :

```
>>> from os import *
```

```
>>> help(open)
```

```
Help on built-in function open in module nt:
```

```
open(...)
```

```
open(path, flags, mode=0o777, *, dir_fd=None)
```

```
Open a file for low level IO. Returns a file handle (integer).
```

```
If dir_fd is not None, it should be a file descriptor open to a directory,  
and path should be relative; path will then be relative to that directory.  
dir_fd may not be implemented on your platform.  
If it is unavailable, using it will raise a NotImplementedError.
```

Dans ce cas un appel comme ci-dessous provoque une erreur de type de variables :

```
21 # Ouverture du fichier  
22 fich=open(nom_fichier,"rb")
```

```
>>> (executing lines 1 to 36 of "test_lecture_fichier_int.py")
```

```
Traceback (most recent call last):
```

```
File "P:\PRO\USB\NSI\NSI STEGANOGRAPHIE\test_lecture_fichier_int  
.py", line 22, in <module>
```

```
fich=open(nom_fichier,"rb")
```

```
TypeError: an integer is required (got type str)
```

