

# M4202C - COMPLÉMENTS D'ALGÈBRE LINÉAIRE

## Premier pas vers la compression d'image

2017/2018 - A. RIDARD





# Table des matières

<b>1</b>	<b>Une (courte) introduction à Python</b>	<b>5</b>
I.	Pour bien commencer . . . . .	5
1.	Types de base . . . . .	5
2.	Listes (un premier conteneur) . . . . .	6
3.	Structure conditionnelle : SI . . . . .	6
4.	Boucles : POUR et TANT QUE . . . . .	6
5.	Chaînes de caractères (un deuxième conteneur) . . . . .	7
6.	Fichiers textes . . . . .	8
7.	Gestion des répertoires . . . . .	9
II.	Des modules utiles pour le traitement d'image . . . . .	10
1.	Tableaux numpy . . . . .	10
2.	Deux autres modules : skimage et pylab . . . . .	12
<b>2</b>	<b>Une (courte) introduction aux images</b>	<b>13</b>
I.	Images noir et blanc . . . . .	13
1.	Définition et résolution . . . . .	13
2.	Codage . . . . .	13
3.	Premiers traitements . . . . .	15
II.	Images couleur . . . . .	17
1.	Avec une palette . . . . .	17
2.	Avec les trois canaux RVB . . . . .	17



# Chapitre 1

## Une (courte) introduction à Python

### Sommaire

<b>I. Pour bien commencer</b>	<b>5</b>
1. Types de base	5
2. Listes (un premier conteneur)	6
3. Structure conditionnelle : SI	6
4. Boucles : POUR et TANT QUE	6
5. Chaînes de caractères (un deuxième conteneur)	7
6. Fichiers textes	8
7. Gestion des répertoires	9
<b>II. Des modules utiles pour le traitement d'image</b>	<b>10</b>
1. Tableaux numpy	10
2. Deux autres modules : skimage et pylab	12

### I. Pour bien commencer

#### 1. Types de base

Les types de base sont :

- les **entiers** (int) : 421, 0, -192
- les **flottants** (float) : 3.14, 0.0, -1.7e-6
- les **booléens** (bool) : True et False

Les opérateurs numériques (applicables aux entiers et aux flottants) sont :

#### Python (Opérateurs numériques).

+, -, *	addition, soustraction, multiplication
/	division (le résultat est un flottant)
**	exponentiation
//, %	quotient, reste de la division euclidienne

Les opérateurs booléens sont :

#### Python (Opérateurs booléens).

a and b	conjonction
a or b	disjonction
not a	négation
in	test de présence
<, >, <=, >=, ==, !=	comparateurs

## 2. Listes (un premier conteneur)

Une liste est une séquence (ordonnée) de valeurs pouvant être de types différents et accessibles par leur index qui **commence à 0** :

### Python (Indexation des séquences).

*Pour les listes, tuples, chaînes de caractères, ...*

<code>len(seq)</code>	longueur de seq
<code>seq[i]</code>	l'élément d'index i de seq
<code>seq[0]</code>	le premier élément de seq
<code>seq[len(seq)-1]</code> ou <code>seq[-1]</code>	le dernier élément de seq
<code>seq[a : b : p]</code>	les éléments d'index a, a+p, ..., a+kp < b

Une liste fait partie des conteneurs <sup>[1]</sup> modifiables auxquels on peut appliquer les **fonctions/méthodes** <sup>[2]</sup> suivantes :

### Python (Opérations sur les conteneurs).

<code>min(cont)</code> , <code>max(cont)</code>	minimum , maximum de cont
<code>sum(cont)</code>	somme des éléments de cont
<code>+</code> , <code>*</code>	concaténation , duplication
<code>cont.count(val)</code>	nombre d'occurrences de val dans cont

*Opérations spécifiques aux listes*

<code>lst.insert(idk, val)</code>	insertion de val dans lst à la position idk
<code>del(lst[idk])</code>	suppression dans lst de l'élément d'index idk

## 3. Structure conditionnelle : SI

La structure conditionnelle permet à un bloc d'instructions de n'être exécuté que si une condition prédéterminée est réalisée. Dans le cas contraire, les instructions sont ignorées et la séquence d'exécution continue à partir de l'instruction qui suit immédiatement la fin du bloc.

### Python (Structure conditionnelle).

<b>if</b> <i>b1</i> :	
<i>bloc d'instructions si b1 est vrai</i>	Plusieurs
<b>elif</b> <i>b2</i> :	elif
<i>bloc d'instructions si b1 est faux et b2 vrai</i>	possibles
<b>else</b> :	
<i>bloc d'instructions sinon</i>	

## 4. Boucles : POUR et TANT QUE

### Boucle itérative : POUR

La boucle itérative sert à répéter un bloc d'instructions un nombre prédéfini de fois.

### Python (Boucle itérative).

<code>for i in range(<sup>0 par défaut</sup>[<i>a</i>, <sup>1 par défaut</sup><i>b</i>], <sup>exclu</sup>[<i>p</i>]) :</code>	i varie de a à b exclu avec un pas de p
<i>bloc d'instructions</i>	

[1]. On en verra d'autres : les tuples, chaînes de caractères et dictionnaires

[2]. La distinction entre fonctions et méthodes sera faite lorsque l'on introduira la Programmation Orientée Objet

## Extension du for

Dans une boucle itérative, la variable d'itération peut prendre ses valeurs dans un conteneur.

### Python (Extension du for).

```
for i in cont :           i prend ses valeurs
    |bloc d'instructions   dans cont
```

## Boucle conditionnelle : TANT QUE

La boucle conditionnelle sert à répéter un bloc d'instructions tant qu'une certaine condition est réalisée.

### Python (Boucle conditionnelle).

```
while b :                 instructions exécutées
    |bloc d'instructions   tant que b est vrai
```

## 5. Chaînes de caractères (un deuxième conteneur)

Comme une liste, une chaîne de caractères est une séquence<sup>[3]</sup> (ordonnée); les caractères étant accessibles par leur index qui commence à 0.

Exemples :

- "" (chaîne vide)
- 'Hei!' (tous les caractères sont autorisés)
- "L'école HEI" (pour considérer l'apostrophe ', la chaîne doit être délimitée par ")

### Une chaîne de caractères n'est pas modifiable

```
>>> ch='HEC'
>>> ch[2]='I'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Les **fonctions**/**méthodes** spécifiques aux chaînes de caractères sont :

### Python (Chaines de caractères).

```
var = eval(ch)           conversion d'une chaîne de caractères
                           en entier, flottant, liste, ...
ch = str(var)             conversion d'un entier, flottant, liste, ...
                           en chaîne de caractères

" - ".join(['Jean','Pierre'])    donne 'Jean-Pierre'
   chaîne de jointure      séquence de chaînes

"2;5.1;12".split(';')           donne ['2','5.1','12']
                           chaîne de séparation
```

### Conversions


```
>>> eval('421')
421
>>> eval('[421,3.14,True]')
[421, 3.14, True]
>>> str(True)
'True'
```

[3]. L'encadré **Indexation des séquences** est donc valable pour les chaînes de caractères

## 6. Fichiers textes

### Des caractères particuliers

- \n permet de faire un saut de ligne
- \t permet de faire une tabulation
- \" permet de mettre \" dans une chaîne délimitée par \"
- \' permet de mettre \' dans une chaîne délimitée par \'



```
>>> ch1="Voici une chaîne de caractères \nqui s'affichera sur deux lignes."
>>> ch1
"Voici une chaîne de caractères \nqui s'affichera sur deux lignes."
>>> print(ch1)
Voici une chaîne de caractères
qui s'affichera sur deux lignes.
>>> ch2="Voici une chaîne avec une\ttabulation"
>>> ch3="Le caractère \\ s'appelle antislash ou backslash pour les anglophones"
>>> ch4="Voici des \" dans une chaîne délimitée par des \""
>>> ch5='Voici des \' dans une chaîne délimitée par des \''
>>> print(ch2)
Voici une chaîne avec une      tabulation
>>> print(ch3)
Le caractère \ s'appelle antislash ou backslash pour les anglophones
>>> print(ch4)
Voici des " dans une chaîne délimitée par des "
>>> print(ch5)
Voici des ' dans une chaîne délimitée par des '
```

### Lecture - Ecriture - Ajout

#### Python (Fichiers).

<code>f=open('chemin\ nomfichier.txt', 'w' )</code>	ouverture en	<code>'r'</code> lecture <code>'w'</code> écriture <code>'a'</code> ajout
<code>f.readline()</code>	lecture d'une ligne (caractère de fin de ligne compris) à partir de l'endroit où l'on se trouve dans le fichier	
<code>f.readlines()</code>	séquence des lignes du fichier	
<code>f.write('texte à écrire')</code>	écriture à partir de l'endroit où l'on se trouve dans le fichier	
<code>f.close()</code>	fermeture	

### Lecture avec readline

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Lecture.txt','r')
>>> f.readline()
'Pour lire ce texte avec Pyhon,\n'
>>> f.readline()
'il existe différentes méthodes'
>>> f.readline()
''
>>> f.close()
```



- Ne pas oublier le \\ dans le chemin
- Le fichier doit exister



## Lecture avec readlines

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Lecture.txt','r')
>>> lst=f.readlines()
>>> lst
['Pour lire ce texte avec Python,\n', 'il existe différentes méthodes']
>>> x=''
>>> for ligne in lst:
...     x=x+ligne
...
>>> x
'Pour lire ce texte avec Python,\nil existe différentes méthodes'
>>> print(x)
Pour lire ce texte avec Python,
il existe différentes méthodes
>>> f.close()
```

## Ecriture

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Ecriture.txt','w')
>>> f.write('Ce texte sera écrit\nsur deux lignes')
35
>>> f.close()
```



| Si le fichier existe, son contenu est effacé, sinon le fichier est créé

## Ajout (en fin de fichier)

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Ecriture.txt','a')
>>> f.write('Attention au retour à la ligne')
30
>>> f.close()
```

## 7. Gestion des répertoires

### Module (os : gestion des répertoires).

import os	Les fonctions seront préfixées par os.
os.getcwd()	Quel est le répertoire de travail ?
os.chdir('chemin')	Changement du répertoire de travail
os.listdir()	Liste des fichiers dans le répertoire de travail

```
>>> import os
>>> os.getcwd()
'C:\\Users\\aridard2\\Documents\\WinPython-32bit-3.3.3.3\\python-3.3.3\\Scripts'
>>> os.chdir('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info')
>>> f=open('Ecriture.txt','r')
>>> lst=f.readlines()
>>> x=''
>>> for ligne in lst:
...     x=x+ligne
...
>>> print(x)
Ce texte sera écrit
sur deux lignesAttention au retour à la ligne
>>> f.close()
```

## II. Des modules utiles pour le traitement d'image

### 1. Tableaux numpy

**Module (numpy : gestion des tableaux).**

```
import numpy as np          Les fonctions seront préfixées par np.
np.array(liste)             Création d'un tableau à partir d'une liste
np.linspace(min,max],nbPoints)  Création d'un tableau 1D
np.arange(min,max[,pas)      Création d'un tableau 1D
np.fromfunction(fct,(nbLig,nbCol))  Création d'un tableau 2D
np.where(condition,val si vrai, val sinon)  Modification d'un
tableau
tableau.reshape(nbElt sur axe 0,...)  Permet de reformer un
tableau
tableau.shape               Nombre d'éléments sur chaque axe
tableau.dtype               Type commun des éléments d'un tableau
```

**Les opérations et les fonctions s'appliquent terme à terme**

Dans les exemples de cette section, le module numpy a déjà été importé avec l'instruction *import numpy as np*.

#### Construction à partir d'une liste

```
>>> tab1=np.array([1.,2.,3.])
>>> tab1.shape
(3,)
>>> tab1.dtype
dtype('float64')
>>> tab2=np.array([[1,2,3],[4,5,6]])
>>> tab2.shape
(2, 3)
>>> tab2.dtype
dtype('int32')
```



| Contrairement aux listes, tous les éléments doivent être du même type

#### Attention

```
>>> np.array([421,3.14,True])
array([ 421. ,   3.14,   1.  ])
>>> np.array([421,3.14,'True'])
array(['421', '3.14', 'True'],
      dtype='<U4')
```

#### Construction à partir d'une fonction

```
>>> def fct(i,j):
...     return(j>=i)
...
>>> np.fromfunction(fct,(3,3))
array([[ True,  True,  True],
       [False,  True,  True],
       [False, False,  True]], dtype=bool)
```

## Construction à l'aide de arange ou linspace (1D)

```
>>> tab1=np.arange(0,12,2)
>>> tab1
array([ 0,  2,  4,  6,  8, 10])
>>> tab1.dtype
dtype('int32')
>>> tab2=np.linspace(0,10,6)
>>> tab2
array([ 0.,  2.,  4.,  6.,  8., 10.])
>>> tab2.dtype
dtype('float64')
```



- range et arange ne génèrent pas des objets de même type (classe)
- Contrairement à range, arange autorise les flottants

## Attention

```
>>> type(range(0,12,2))
<class 'range'>
>>> type(np.arange(0,12,2))
<class 'numpy.ndarray'>
>>> np.arange(0,3,0.5)
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5])
>>> range(0,3,0.5)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
```

## Modification de la forme

```
>>> tab1=np.arange(0,12,2)
>>> tab1
array([ 0,  2,  4,  6,  8, 10])
>>> tab2=tab1.reshape(2,3)
>>> tab2
array([[ 0,  2,  4],
       [ 6,  8, 10]])
```

## Modification multiple du contenu

```
>>> tab=np.array([[1,2,3],[4,5,6]])
>>> tab[1:]=0
>>> tab
array([[1, 2, 3],
       [0, 0, 0]])
```

## Modification en parallèle du contenu

```
>>> tab=np.array([[1,2,3],[4,5,6]])
>>> tab[1:]=[1,2,3]
>>> tab
array([[1, 2, 3],
       [1, 2, 3]])
```

## Modification conditionnelle du contenu

```
>>> tab=np.array([[1,2,3],[4,5,6]])
>>> tab=np.where(tab%2==0,tab,0)
>>> tab
array([[0, 2, 0],
       [4, 0, 6]])
```



Les opérations et les fonctions s'appliquent terme à terme.

## Traitement vectoriel

```
>>> tab=np.arange(1,7).reshape(2,3)
>>> tab
array([[1, 2, 3],
       [4, 5, 6]])
>>> tab%2==1
array([[ True, False,  True],
       [False,  True, False]], dtype=bool)
>>> tab+1
array([[2, 3, 4],
       [5, 6, 7]])
>>> tab**2
array([[ 1,  4,  9],
       [16, 25, 36]])
>>> np.sin(tab)
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ]])
```

## 2. Deux autres modules : skimage et pylab

### Module (skimage : gestion des images).

<code>from skimage import io</code>	importation du sous-module io
<code>io.imread('chemin')</code>	Importation d'une image
<code>io.imsave('nomFichier.png',tab)</code>	Enregistrement d'une image

### Module (pylab : gestion des graphiques).

<code>import pylab as pl</code>	Les fonctions seront préfixées par pl.
<code>pl.imshow(tab,cmap=palette)</code>	Création d'une image
<code>pl.hist(liste,bins=nbClasses)</code>	Création d'un histogramme
<code>pl.plot(listeAbs,listeOrd)</code>	Création d'un graphique
<code>pl.subplot(nbLig,nbCol,num)</code>	Création d'un sous-graphique
<code>pl.figure('nomFenêtre')</code>	Nom de la fenêtre graphique
<code>pl.title('nomGraphique')</code>	Nom du (sous-)graphique
<code>pl.xlabel('nomAbs')</code>	Nom de l'axe des abscisses
<code>pl.ylabel('nomOrd')</code>	Nom de l'axe des ordonnées
<code>pl.xlim(min,max)</code>	Domaine de l'axe des abscisses
<code>pl.ylim(min,max)</code>	Domaine de l'axe des ordonnées
<code>pl.show()</code>	Affichage de la fenêtre graphique
<code>pl.close()</code>	Fermeture de la fenêtre graphique

# Chapitre 2

## Une (courte) introduction aux images

### Sommaire

<b>I. Images noir et blanc</b>	<b>13</b>
1. Définition et résolution	13
2. Codage	13
3. Premiers traitements	15
<b>II. Images couleur</b>	<b>17</b>
1. Avec une palette	17
2. Avec les trois canaux RVB	17

### I. Images noir et blanc

Une image est constituée d'un ensemble de points (pixels) colorés. En codant chaque couleur de chaque pixel, on aboutit alors à des images numériques matricielles.

#### 1. Définition et résolution

La **définition** d'une image est le nombre de pixels qui la composent : nombre de ligne  $\times$  nombre de colonnes.

La **résolution** d'une image est le nombre de points pour une longueur donnée. Elle est exprimée en points par pouce d'acronyme DPI (Dots Per Inch). La longueur donnée est ici le pouce (25,4 mm).

On a donc : **résolution (en DPI)  $\times$  dimension (en pouces) = définition**

La résolution permet donc d'obtenir la grandeur réelle d'une image sur un support physique. En effet, une image de définition 150  $\times$  100 sur un support de résolution 72 DPI (écran d'ordinateur) mesure :

- en pouces : 150/72  $\times$  100/72 soit environ 2,08  $\times$  1,39
- en mm : 150\*25,4 / 72  $\times$  100 \*25,4/ 72 soit environ 52,9  $\times$  35,3

#### 2. Codage

##### Un pixel est un entier

Une image numérique (matricielle) en niveaux de gris est un tableau de valeurs souvent codées sur 8 bits (entiers de 0 à 255).



Valeurs des niveaux de gris et teintes associées

Exemples :

- image **miniPouce** codée sur 8 bits de définition 100 x 100



- image **Camera** codée sur 8 bits de définition 512 x 512



### Une image est un tableau



```
>>> os.chdir('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info')
>>> img=io.imread('camera.png')
>>> type(img)
<class 'numpy.ndarray'>
>>> img.dtype
dtype('uint8')
>>> img.shape
(512, 512)
```



Le pixel de position (0,0) est celui en haut à gauche

(0,0)	(0,1)	...	(0,510)	(0,511)
(1,0)	(1,1)	...	(1,510)	(1,511)
⋮	⋮	⋮	⋮	⋮
(510,0)	(510,1)	...	(510,510)	(510,511)
(511,0)	(511,1)	...	(511,510)	(511,511)

Tableau des positions



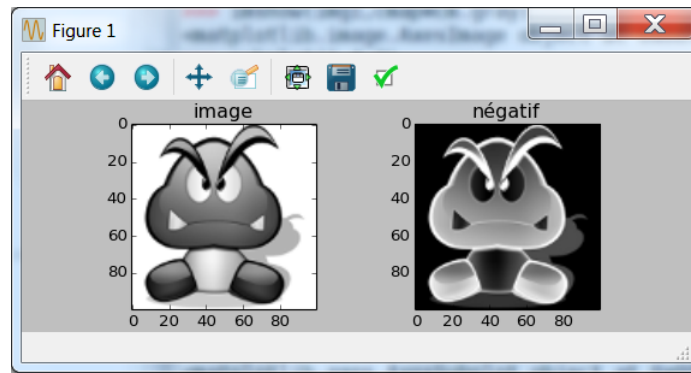
156	157	...	152	152
156	157	...	152	152
⋮	⋮	⋮	⋮	⋮
121	123	...	113	111
121	123	...	113	111

Tableau des valeurs

### 3. Premiers traitements

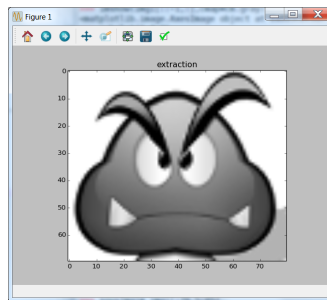
#### Négatif

En changeant val en  $\text{max} - \text{val}$  ( $\text{max} = 255$  donc 0 devient 255 et 255 devient 0), on obtient :



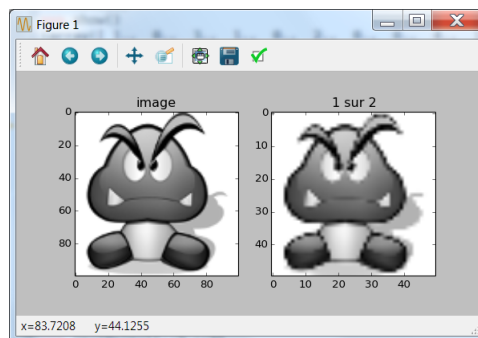
#### Rognage

En prenant une partie du tableau, on obtient :



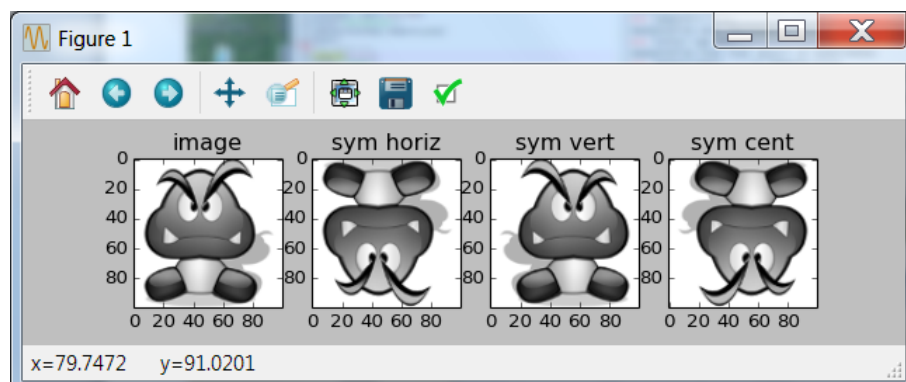
#### Diminution de la définition

En prenant une ligne sur deux et une colonne sur deux, on obtient :



#### Transformations géométriques

Par symétrie, on obtient :



## Posterisation

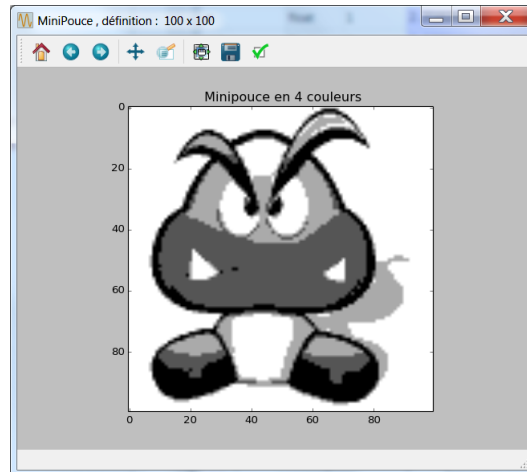
Le principe consiste à diminuer le nombre de tons dans l'image. Le codage peut alors utiliser un format prenant moins de place :

- en 2 niveaux de gris (seuillage), les valeurs sont 0 et 1 : codage sur 1 bit par pixel
- en 4 niveaux de gris, les valeurs sont 0, 1, 2 et 3 : codage sur 2 bits par pixel (00, 01, 10, 11)
- en 8 niveaux de gris, les valeurs sont 0, 1, 2, 3, 4, 5, 6 et 7 : codage sur 3 bits par pixel (000, 001, 010, ..., 111)
- ...

Procédé en 4 niveaux de gris : si  $x$  désigne la valeur du pixel, alors  $x \mapsto$

$$\begin{cases} 0 & \text{si } x < 64 \\ 85 & \text{si } 64 \leq x < 128 \\ 170 & \text{si } 128 \leq x < 192 \\ 255 & \text{si } 192 \leq x \end{cases}$$

Résultat du procédé :



### TP Python

Pour chacun des traitements présentés, déclarer une fonction définie de la manière suivante :

- **Entrée** : une image couleur (tableau 3D)
- **Sortie** : l'image transformée

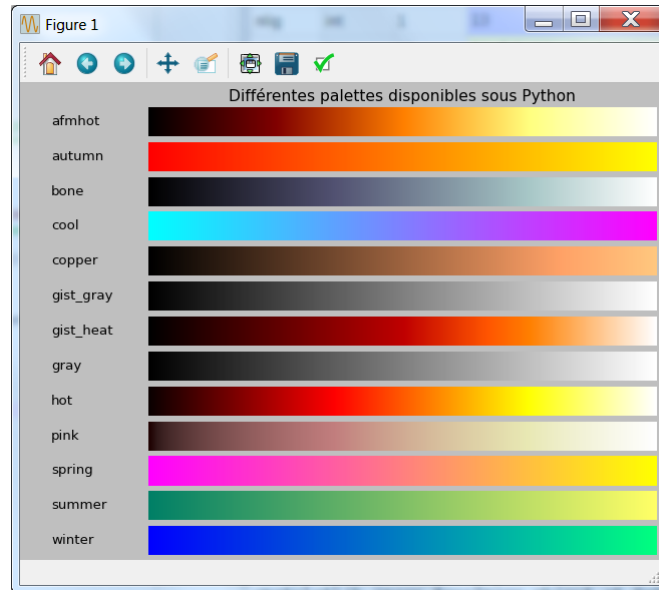


## II. Images couleur

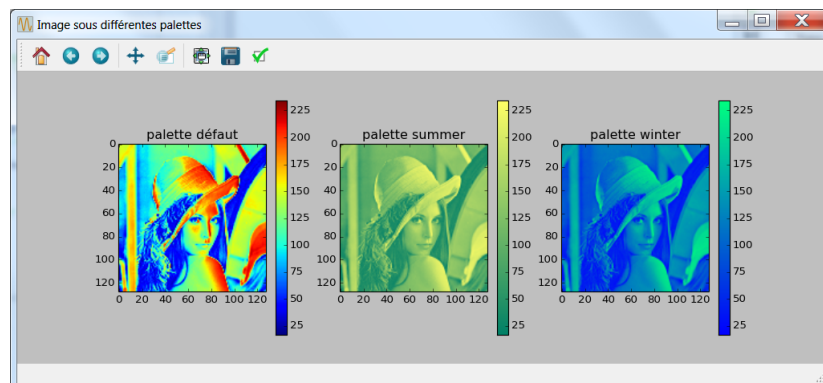
### 1. Avec une palette

Une image numérique colorée **selon une palette** est un tableau d'entiers souvent compris entre 0 et 255 (comme en noir et blanc).

Différentes palettes sont disponibles sous Python :



Exemples d'affichages selon différentes palettes :

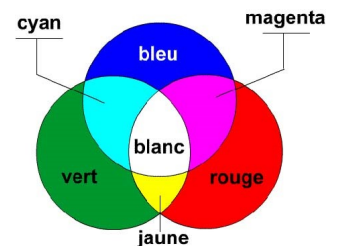


### 2. Avec les trois canaux RVB

Un modèle possible de construction des couleurs est le système additif basé sur les 3 couleurs primaires : rouge, vert et bleu.

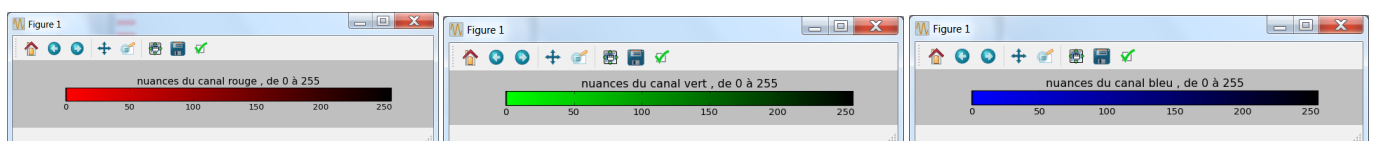
Exemple : cyan = 50% bleu + 50% vert

En jouant sur les pourcentages des couleurs primaires, on obtient un nombre important de couleurs distinctes.



Dans ce modèle, chaque pixel est codé par une liste de trois entiers souvent compris entre 0 et 255 : on parle de canal rouge, canal vert et canal bleu (dans cet ordre). On a ainsi  $256 \times 256 \times 256 = 16\,777\,216$  couleurs!

Voici les nuances des trois canaux :



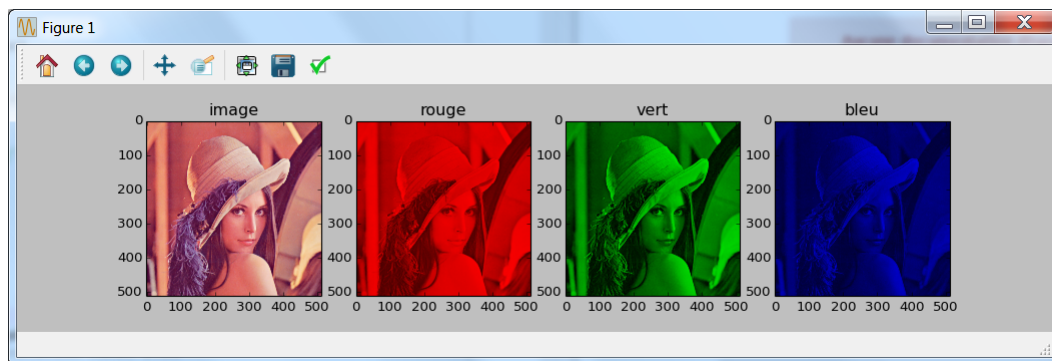
Une image couleur et son tableau de valeurs :



(226,137,125)	(226,137,125)	...	...	...
(230,148,122)	(221,130,110)	...	...	...
⋮	⋮	⋮	⋮	⋮
...	...	...	(181,71,81)	(185,74,81)

Son poids est alors  $512 \times 512 \times 3 \times 8 = 6\,291\,456$  bits soit 786 432 octets.

Ses différentes composantes dans les canaux :



## TP Python

| Masquage d'un texte dans une image couleur.