COURS D'INFORMATIQUE (TRONC COMMUN)

HEI 2 - 2015/2016

D. LOSFELD - C. MAHIEU - A. RIDARD

Table des matières

Période 1

Rappels - Variables locales et globales - Affectation par référencement

I. Quelques rappels

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, chaines de caractères, ...

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

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

Python (Opérations sur les conteneurs).

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

Opérations spécifiques aux listes

lst.insert(idx,val)	insertion de val dans lst à la position idx
del(lst[idx])	suppression dans lst de l'élément d'index idx

3. Les instructions conditionnelles : SI

Les instructions conditionnelles permettent 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.

Sous Python, la syntaxe est la suivante :

Python (Instructions conditionnelles).

if *b1* **:**
    |*bloc d'instructions si b1 est vrai*
elif *b2* **:**
    |*bloc d'instructions si b1 est faux et b2 vrai*
else :
    |*bloc d'instructions sinon*

Plusieurs elif possibles

4. Les boucles : POUR et TANT QUE

La boucle itérative : POUR

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

Sous Python, la syntaxe est la suivante :

Python (Boucle itérative).

for *i* in range($\overbrace{[a,}^{\text{0 par défaut}}$ *b* $\overbrace{[,p]}^{\text{1 par défaut}}$) **:**
    |*bloc d'instructions*

(*b* : exclu)

i varie de a à b exclu avec un pas de p

La boucle conditionnelle : TANT QUE

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

Sous Python, la syntaxe est la suivante :

Python (Boucle conditionnelle).

while *b* **:**
    |*bloc d'instructions*

instructions éxécutées tant que b est vrai

[1]. On en verra d'autres : les tuples, chaînes de caractères et dictionnaires (MP)
[2]. La distinction entre fonctions et méthodes sera faite lorsque l'on introduira la Programmation Orientée Objet

5. tuples (un deuxième conteneur)

Comme une liste, un tuple est une séquence [3] (ordonnée) de valeurs pouvant être de types différents et accessibles par leur index qui **commence à 0**.

Exemples :
- (421,3.14,False) ou 421,3.14,False
- (421,) ⚠ne pas oublier la virgule pour les tuples à un seul élément

⚠ Un tuple fait également partie des conteneurs [4] mais, contrairement aux listes, **les tuples ne sont pas modifiables**.

6. Extension du POUR

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

Sous Python, la syntaxe est :

Python (Extension du for).

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

II. Différentes manières de programmer

Python permet de mettre en œuvre, éventuellement partiellement, les trois types de programmation suivants.

1. Programmation impérative

En programmation, il existe différents paradigmes (modes de pensée) permettant de rédiger des algorithmes. Celui utilisé jusqu'ici est la **programmation impérative** qui décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme. La plupart des langages impératifs comporte trois types d'instructions principales :
- l'assignation ou affectation
- les instructions conditionnelles
- les boucles (itératives et conditionnelles)

On peut modéliser un tel programme par une **machine à états** qui représente les états successifs de la mémoire. Cela nécessite pour le programmeur de connaître, à tout instant, l'état exact de la mémoire que le programme modifie. Pour effectuer ce suivi, on pourra d'ailleurs utiliser, sous Python, l'**explorateur de variables**.

Cependant, il arrive que la conception choisie par le programmeur l'amène à mettre à jour des variables involontairement. Ces modifications «périphériques», désignées sous le terme générique d'**effets de bord**, compliquent grandement la compréhension des programmes et sont la source de nombreuses difficultés et de bugs.

Exemple Python (Effet de bord).

```
>>> def f(x):
...     return(x+a)
...
>>> a=5
>>> f(10)
15
>>> a=10
>>> f(10)
20
```

2. Programmation fonctionnelle

La **programmation fonctionnelle** s'affranchit de façon radicale des effets de bord en interdisant toute opération d'affectation. Elle considère le calcul en tant qu'évaluation de fonctions mathématiques et rejette le changement d'état.

[3]. L'encadré **Indexation des séquences** est donc valable pour les tuples
[4]. L'encadré **Opérations sur les conteneurs** est donc valable pour les tuples à l'exception des opérations spécifiques aux listes

3. Programmation Orientée Objet (POO)

La **programmation orientée objet** quant à elle permet de regrouper les données et les fonctions agissant sur ces données afin d'isoler le plus possible la portée des variables.

L'intégration des données et des fonctions se réalise par un nouveau "type" appelé **Classe**. Python est (depuis la version 3) "développé" en modèle Objet : tout type est une classe que l'on peut visualiser par la fonction type().

Exemple Python (Classes).

```
>>> type(421)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> type(True)
<class 'bool'>
>>> type([421,3.14,True])
<class 'list'>
>>> type((421,3.14,True))
<class 'tuple'>
>>> type(2+3j)
<class 'complex'>
```

Ces classes comportent non seulement des données appelées **attributs** mais aussi des fonctions appelées **méthodes** permettant de répondre à des « messages » et ainsi de manipuler les données.

A partir de ces classes, on crée (instanciation) des objets référencés par des variables.

Exemple Python (Instanciation).

```
>>> cplxe=2+3j
>>> type(cplxe)
<class 'complex'>
```

On dit que cplxe est une référence à un objet instancié à partir de la classe complex.

Sous Python, la syntaxe est :
- nomObjet.nomAttribut pour accéder aux attributs d'un objet
- nomObjet.nomMethode(paramètres) pour utiliser une méthode d'un objet

Exemple Python (Attributs et méthodes).

```
>>> cplxe=2+3j
>>> cplxe.real
2.0
>>> cplxe.imag
3.0
>>> cplxe.conjugate()
(2-3j)
>>> cplxe.__add__(1+2j)
(3+5j)
```

Pour obtenir de l'aide sur les attributs et les méthodes d'un objet, on pourra utiliser help(nomObjet).

III. Variables locales et globales

Pour réduire les effets de bord, on peut utiliser les variables locales : leur portée est strictement limitée à la fonction où elles sont utilisées car elles sont désallouées immédiatement en sortie de fonction via un « ramasse-miettes ».

Exemple Python (Variables locales).

```
>>> def f(x):
...     b=20
...     res=x+a+b
...     return(res)
...
>>> a,b=5,0
>>> f(10)
35
>>> a,b
(5,0)
>>> x
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> res
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'res' is not defined
```

Dans une définition de fonction, il est possible de déclarer des variables globales (dangereux).

Exemple Python (Variables globales).

```
>>> def f(x):
...     global b
...     b=20
...     res=x+a+b
...     return(res)
...
>>> a=5
>>> f(10)
35
>>> a
5
>>> x
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> b
20
>>> res
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'res' is not defined
```

IV. Affectation par référencement

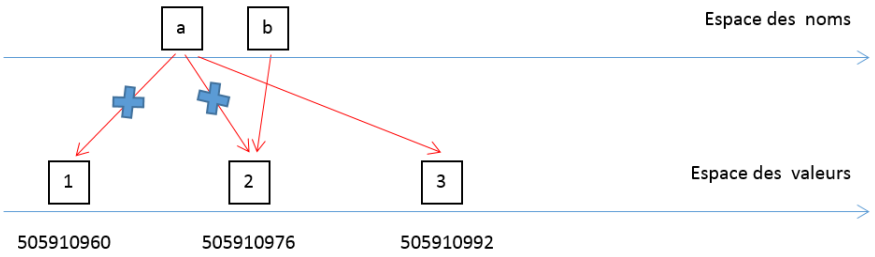
1. Principe

Lors d'une affectation sous Python, le nom de la variable est enregistré dans l'espace des noms et pointe vers (ou référence) la valeur mémorisée à une autre adresse que l'on peut obtenir avec la fonction id(nomVar).

Exemple Python (Affectation par référencement).

```
>>> a=1
>>> id(a)
505910960
>>> a=2
>>> id(a)
505910976
>>> b=a
>>> id(b)
505910976
>>> a=3
>>> id(a)
505910992
>>> id(b)
505910976
>>> id(3)
505910992
>>> id(2)
505910976
```

⚠ Lorsque la valeur de a change, celle de b ne change pas



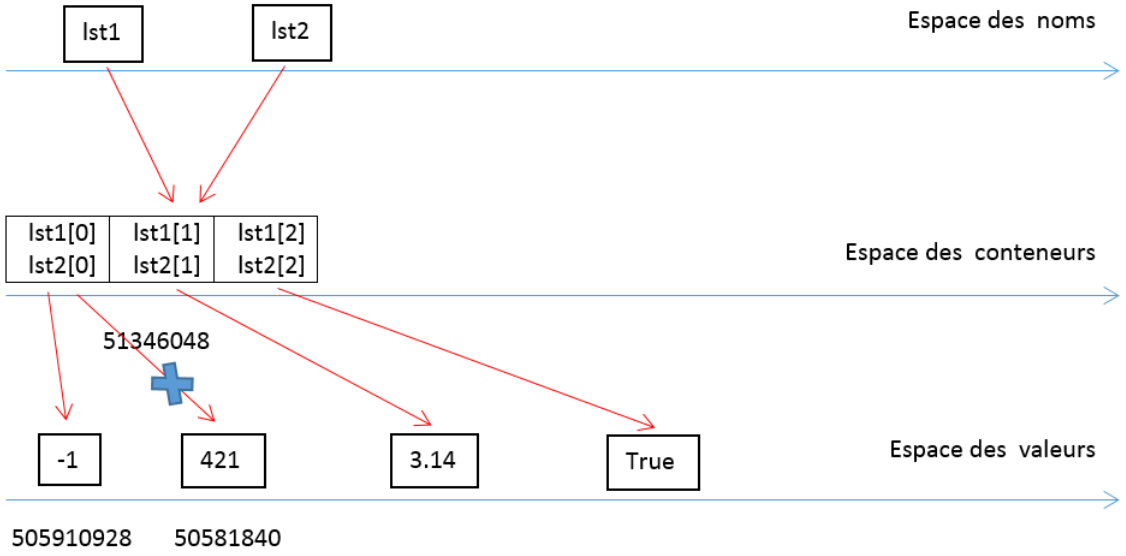
2. Cas des listes

Danger

Exemple Python (Affectation de liste 1).

```
>>> lst1=[421,3.14,True]
>>> lst2=lst1
>>> id(lst1)
51346048
>>> id(lst2)
51346048
>>> id(lst2[0])
50581840
>>> id(lst1[0])
50581840
>>> lst1[0]=-1
>>> lst1
[-1, 3.14, True]
>>> lst2
[-1, 3.14, True]
>>> id(lst1)
51346048
>>> id(lst2)
51346048
>>> id(lst1[0])
505910928
>>> id(lst2[0])
505910928
```

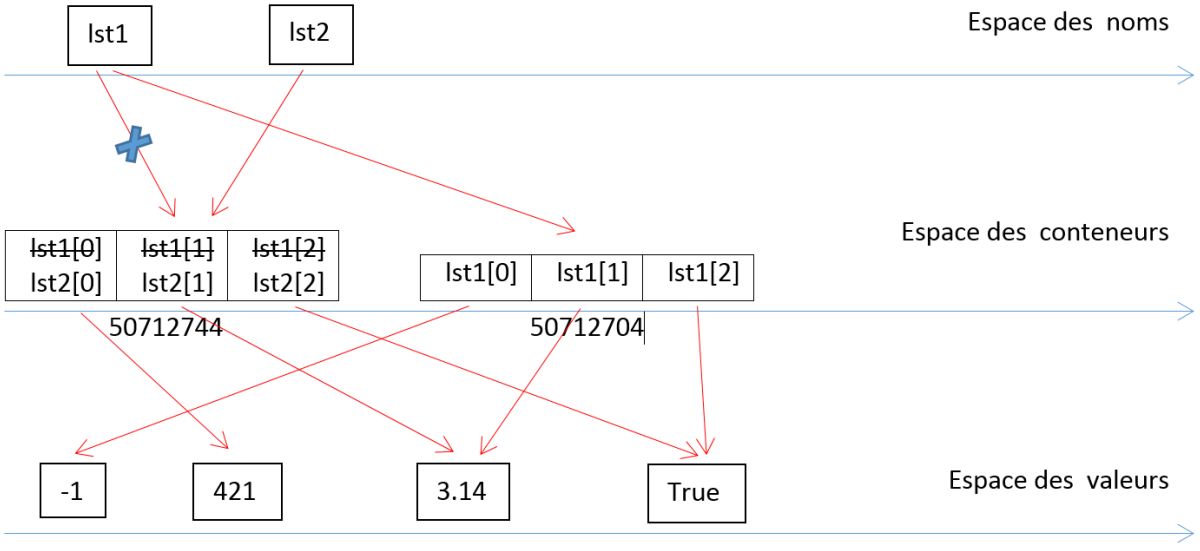
⚠ Lorsque la valeur de lst1[0] change, celle de lst2[0] change aussi



Exemple Python (Affectation de liste 2)**.**

```
>>> lst1=[421,3.14,True]
>>> lst2=lst1
>>> lst1=[-1,3.14,True]
>>> lst1
[-1, 3.14, True]
>>> lst2
[421, 3.14, True]
>>> id(lst1)
50712744
>>> id(lst2)
50712704
```

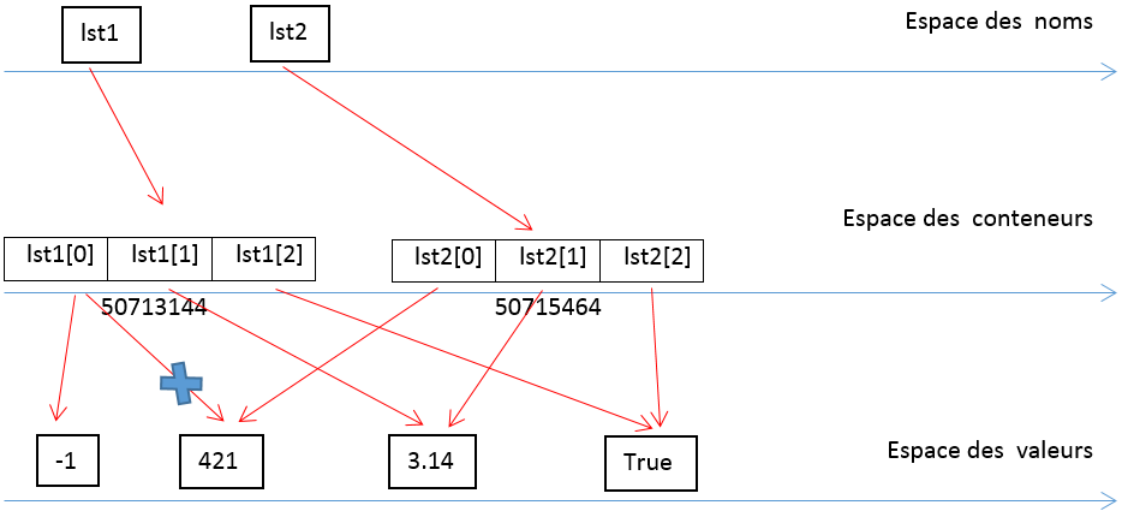
⚠ Lorsque la valeur de lst1 change, celle de lst2 ne change pas



Copie par concaténation

Exemple Python (Copie par concaténation).

```
>>> lst1=[421,3.14,True]
>>> lst2=lst1+[]
>>> lst1[0]=-1
>>> lst1
[-1, 3.14, True]
>>> lst2
[421, 3.14, True]
>>> id(lst1)
50713144
>>> id(lst2)
50715464
```



Copie par duplication

Exemple Python (Copie par duplication).

```
>>> lst1=[421,3.14,True]
>>> lst2=lst1*1
>>> lst1[0]=-1
>>> lst1
[-1, 3.14, True]
>>> lst2
[421, 3.14, True]
>>> id(lst1)
50715664
>>> id(lst2)
50715624
```

Copie par tranchage

Exemple Python (Copie par tranchage)**.**

```
>>> lst1=[421,3.14,True]
>>> lst2=lst1[:]
>>> lst1[0]=-1
>>> lst1
[-1, 3.14, True]
>>> lst2
[421, 3.14, True]
>>> id(lst1)
50715464
>>> id(lst2)
50715664
```

Module copy

Exemple Python (Copie avec la fonction copy)**.**

```
>>> from copy import copy
>>> lst1=[421,3.14,True]
>>> lst2=copy(lst1)
>>> lst1[0]=-1
>>> lst1
[-1, 3.14, True]
>>> lst2
[421, 3.14, True]
>>> id(lst1)
50715464
>>> id(lst2)
39252696
```

Exemple Python (Comparaison des fonctions copy et deepcopy)**.**

```
>>> from copy import copy,deepcopy
>>> lst1=[[421,3.14,True],[-421,-3.14,False]]
>>> lst2=copy(lst1)
>>> lst3=deepcopy(lst1)
>>> lst1[0][0]=-1
>>> lst1
[[-1, 3.14, True], [-421, -3.14, False]]
>>> lst2
[[-1, 3.14, True], [-421, -3.14, False]]
>>> lst3
[[421, 3.14, True], [-421, -3.14, False]]
>>> id(lst1[0])
51276256
>>> id(lst2[0])
51276256
>>> id(lst3[0])
51276336
```


La fonction copy n'est pas adaptée aux listes de listes. On utilisera donc systématiquement la fonction deepcopy.

Exercice : représenter, à l'aide d'un schéma, l'exemple précédent (on pourra introduire un deuxième espace de conteneurs).

Période 2

Chaînes de caractères - Fichiers textes - Tableaux numpy - Graphes

I. Chaînes de caractères (un troisième conteneur)

1. Définition

Comme une liste, une chaîne de caractères est une séquence [1] (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 fait également partie des conteneurs [2] mais, contrairement aux listes, **les chaînes de caractères ne sont pas modifiables**.

Exemple Python (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 chaine de caractères en entier, flottant, liste, ...
ch = str(var)	conversion d'un entier, flottant, liste, ... en chaine de caractères
" − ".join(['Jean','Pierre'])	donne 'Jean−Pierre'
"2 ;5.1 ;12" . split(';')	donne ['2','5.1','12']

chaine de jointure — séquence de chaines — chaine de séparation

[1]. L'encadré **Indexation des séquences** est donc valable pour les chaînes de caractères
[2]. L'encadré **Opérations sur les conteneurs** est donc valable pour les chaînes de caractères à l'exception des opérations spécifiques aux listes

Exemple Python (Conversions).

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

II. Fichiers textes

1. 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 '

Exemple Python (Caractères particuliers 1).

```
>>> 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.
```

Exemple Python (Caractères particuliers 2).

```
>>> 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 '
```

2. Lecture - Ecriture - Ajout

Python (Fichiers).

f=open('chemin\ nomfichier.txt' , 'r' / 'w' / 'a') ouverture en lecture / écriture / ajout

f.readline() lecture d'une ligne (caractère de fin de ligne compris) à partir de l'endroit où l'on se trouve dans le fichier

f.readlines() séquence des lignes du fichier

f.write('texte à écrire') écriture à partir de l'endroit où l'on se trouve dans le fichier

f.close() fermeture

Lecture

Exemple Python (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

Exemple Python (Attention).

```
>>> f=open('C:\Users\aridard2\Desktop\2015_2016\Info\Lecture.txt','r')
File "<stdin>", line 1
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: ...
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Lectures.txt','r')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'C:\\Users\\aridard2\\Desktop\\...
```

Exemple Python (Lecture avec readlines).

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

Ecriture

Exemple Python (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

Exemple Python (Ajout).

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

⚠ Cet ajout se fait en fin de fichier

III. 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

Exemple Python (Gestion des répertoires).

```
>>> 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()
```

IV. 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(nbElts 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.*

1. Construction

A partir d'une liste

Exemple Python (A 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

Exemple Python (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')
```

A partir d'une fonction

Exemple Python (A 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)
```

A l'aide de arange ou linspace (1D)

Exemple Python (A l'aide de arange ou linspace).

```
>>> 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

Exemple Python (range & arange).

```
>>> 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
```

2. Modification

Forme

Exemple Python (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]])
```

Contenu

Exemple Python (Modification multiple).

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

Exemple Python (Modification en parallèle).

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

Exemple Python (Modification conditionnelle).

```
>>> 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]])
```

3. Traitement vectoriel

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

Exemple Python (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 ]])
```

V. Graphes

1. Définition et exemples

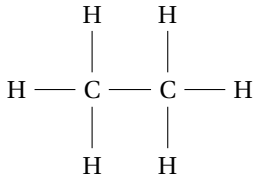
Un graphe est un couple formé de deux ensembles :

- les sommets (ou noeuds) du graphe
- les arêtes du graphe

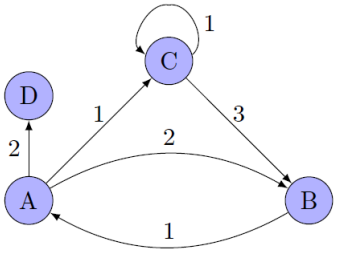
Un graphe peut être orienté ou non, pondéré ou non.

Exemples :

- Graphe non orienté et non pondéré



- Graphe orienté et pondéré



2. Représentation par matrice d'adjacence

On considère :

- l'ensemble S des sommets que l'on notera $1, 2, \ldots, n$.
- l'ensemble A des arêtes orientées et pondérées que l'on notera (i, j, p_{ij}) avec i, j dans S et p_{ij} le poids de i vers j.

Définition (Matrice d'adjacence)**.**

La matrice d'adjacence (m_{ij}) du graphe orienté et pondéré (S, A) est définie par :

$$m_{ij} = \begin{cases} p_{ij} & \text{si } (i, j, p_{ij}) \in A \\ 0 & \text{sinon} \end{cases}$$

Remarques :
- si le graphe n'est pas orienté, la matrice est symétrique
- si le graphe n'est pas pondéré, $p_{ij} = 1$

Exemple : la matrice d'adjacence du graphe orienté et pondéré de l'exemple précédent est $\begin{pmatrix} 0 & 2 & 1 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Période 3

Images - Graphiques

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

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

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

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

Dans les exemples de ce chapitre, les modules os, numpy, skimage et pylab ont déjà été importés.

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 × 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)*** × ***dimension (en pouces) = définition***

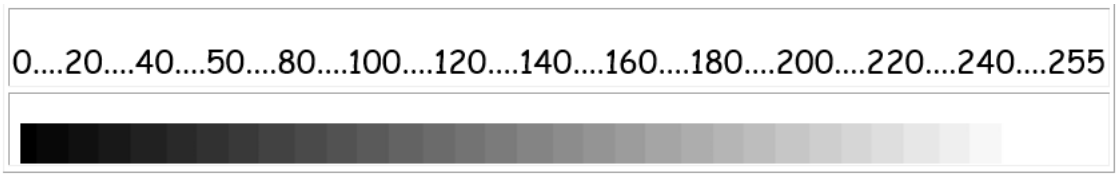
La résolution permet donc d'obtenir la grandeur réelle d'une image sur un support physique. En effet, une image de définition 150 × 100 sur un support de résolution 72 DPI (écran d'ordinateur) mesure :

- en pouces : 150/72 × 100/72 soit environ 2,08 × 1,39
- en mm : 150*25,4 /72 × 100 *25,4/ 72 soit environ 52,9 × 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 défnition 100 x 100

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

Une image est un tableau

Exemple Python (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

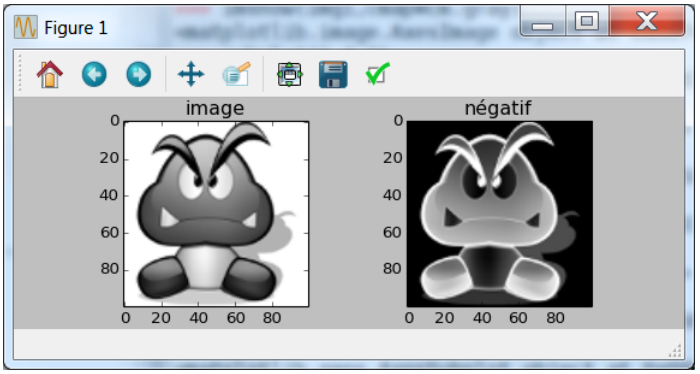
156	157	…	152	152
156	157	…	152	152
⋮	⋮	⋮	⋮	⋮
121	123	…	113	111
121	123	…	113	111

Tableau des valeurs

3. Premiers traitements

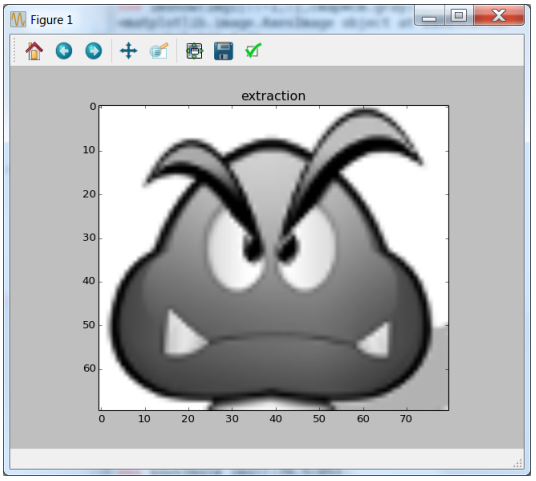
Négatif

En changeant val en max – val (max = 255 donc 0 devient 255 et 255 devient 0), on obtient :



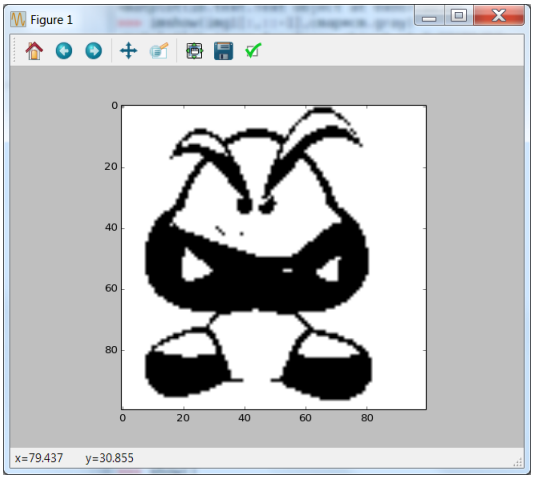
Rognage

En prenant une partie du tableau, on obtient :



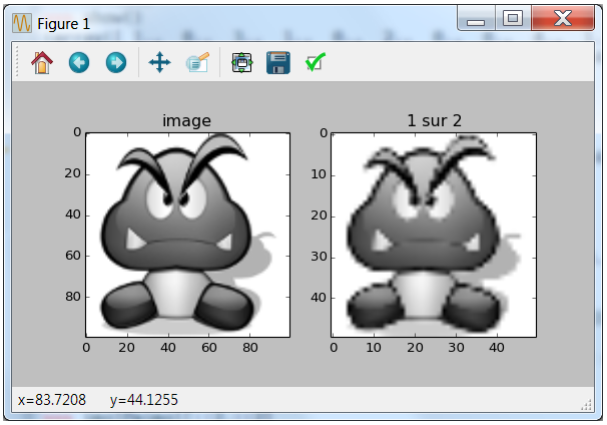
Seuillage

En limitant le nombre de gris à 2 valeurs (si val < valeurChoisie, alors 0 et sinon 255), on obtient :



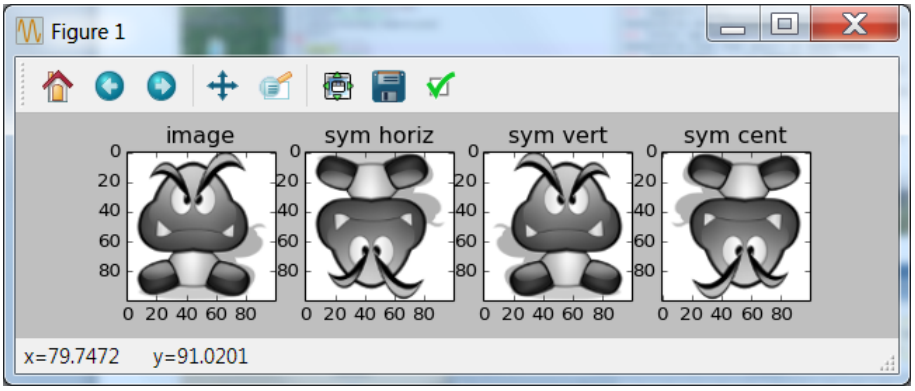
Diminution de la définition

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



Transformations géométriques

En transformant le tableau, on obtient :



4. D'autres traitements

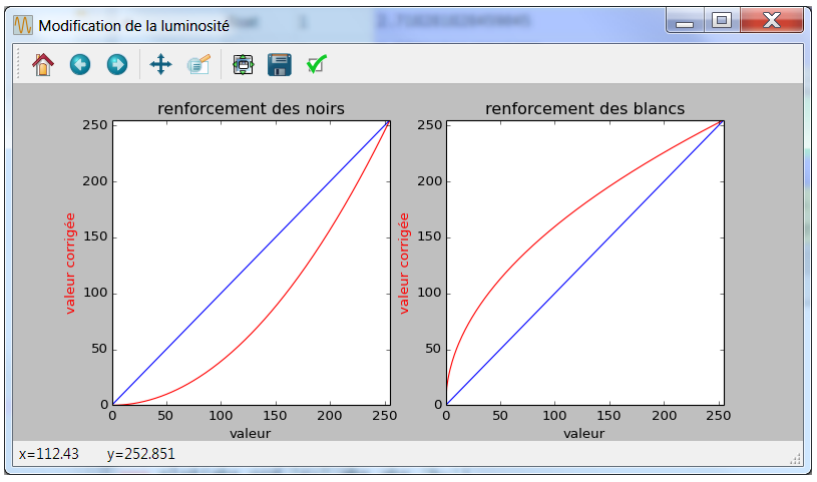
Les traitements suivants opèrent pixel par pixel sans considérer les voisins.

Luminosité

Pour éclaircir (resp. assombrir) une image, il suffit d'augmenter (resp. diminuer) la valeur du codage d'un pixel.

Procédé : si x désigne la valeur du pixel alors :

- $x \mapsto \left\lfloor \frac{x^2}{255} \right\rfloor$ permet d'assombrir l'image où $\left\lfloor \frac{x^2}{255} \right\rfloor$ désigne la partie entière de $\frac{x^2}{255}$
- $x \mapsto \lfloor \sqrt{255x} \rfloor$ permet d'éclaircir l'image

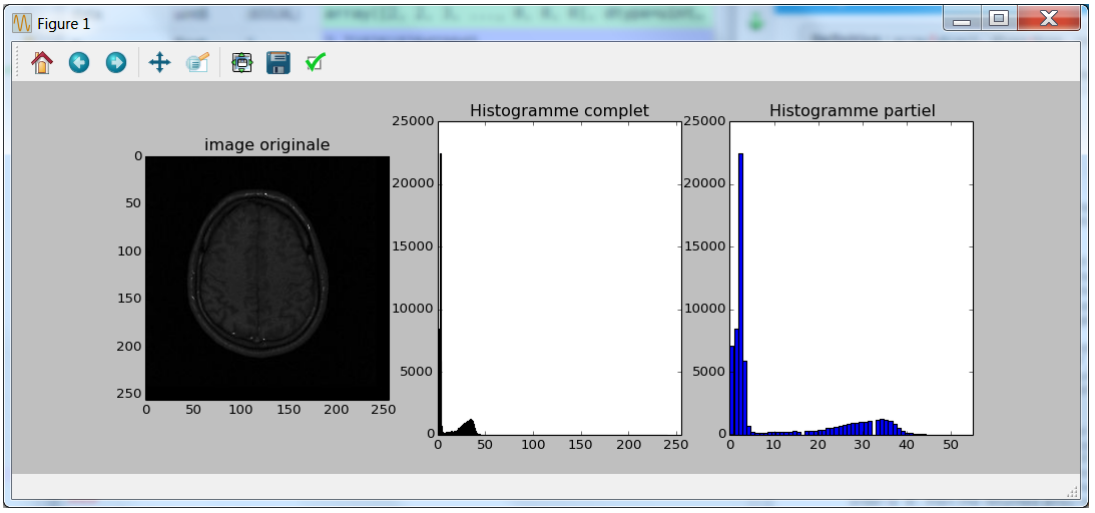


Visualisation de la correction (courbes rouges)

Résultat du procédé :

Contraste

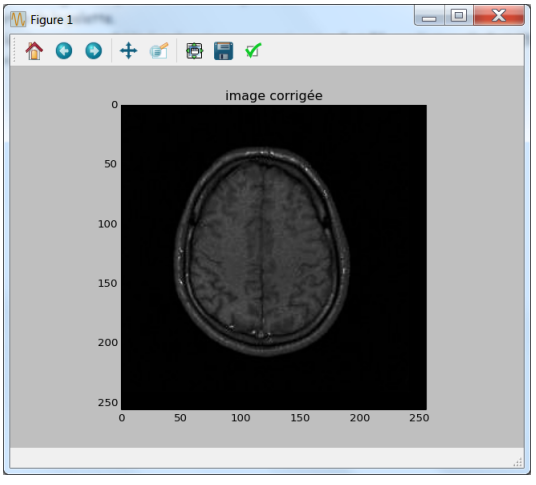
Le rehaussement du contraste peut se réaliser à l'aide de l'histogramme d'une image.
Voici une image représentant une tranche de cerveau humain acquise en Résonance Magnétique Nucléaire (IRM).



On constate sur l'histogramme [1] que les valeurs (tons) sont quasiment toutes entre 0 et 50 (la dynamique de l'image n'occupe qu'une partie de la palette).
Eclaircissons les tons moyens en répartissant les tons compris entre 8 et 50 dans l'intervalle $[8,150[$ et ceux compris entre 50 et 255 dans l'intervalle $[150,255]$.

Procédé : si x désigne la valeur du pixel, alors $x \mapsto \begin{cases} x & \text{si } x < 8 \\ 8 + \frac{150-8}{50-8}(x-8) & \text{si } 8 \leq x < 50 \\ 150 + \frac{255-150}{255-50}(x-50) & \text{si } 50 \leq x \end{cases}$

Résultat du procédé :



[1]. Sa construction sera réalisée plus loin

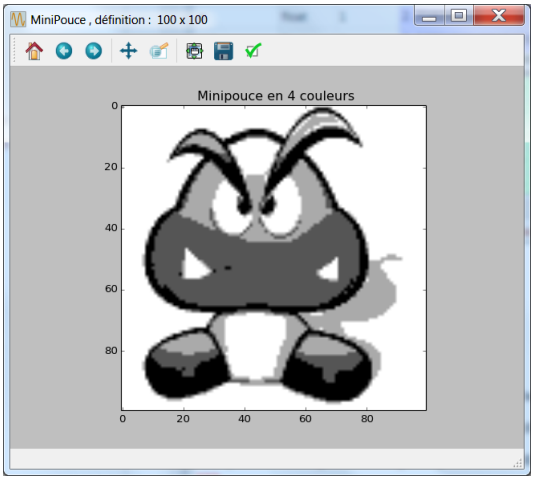
Posterisation

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 :

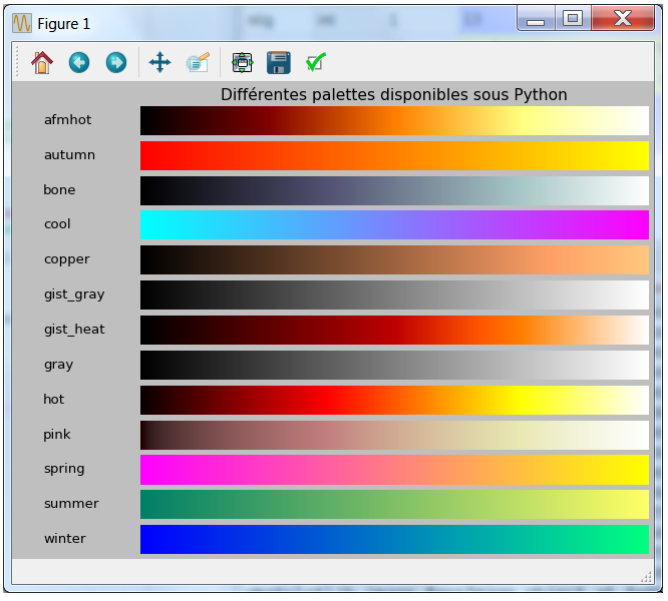


II. Images couleur

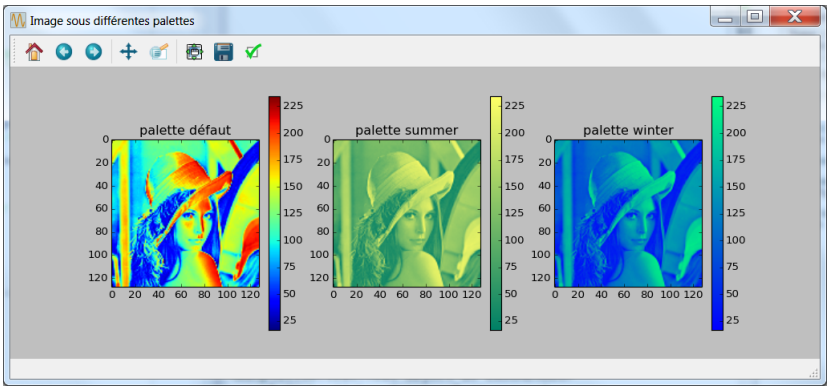
1. Avec une palette

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

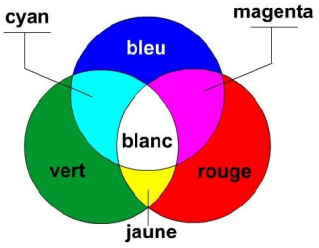
Différentes palettes sont disponibles sous Python :



Exemples d'affichages selon différentes palettes :



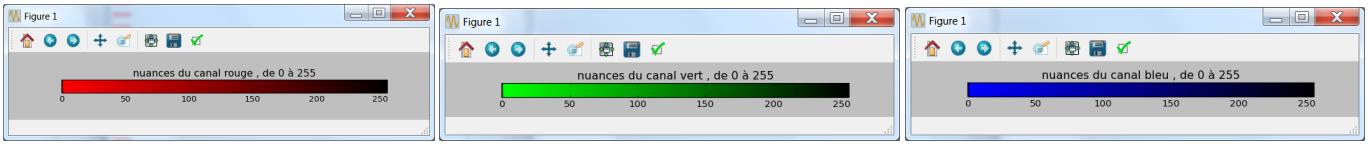
2. Avec les trois canaux RVB



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 :

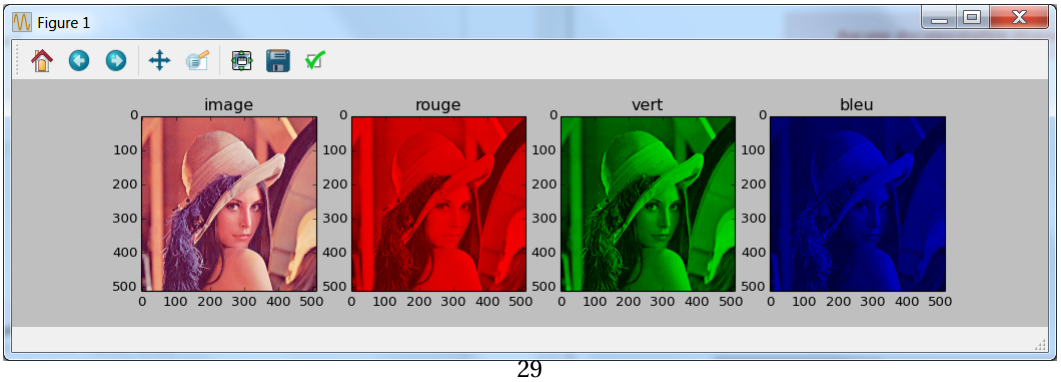


Une image couleur et son tableau de valeurs :

(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 :

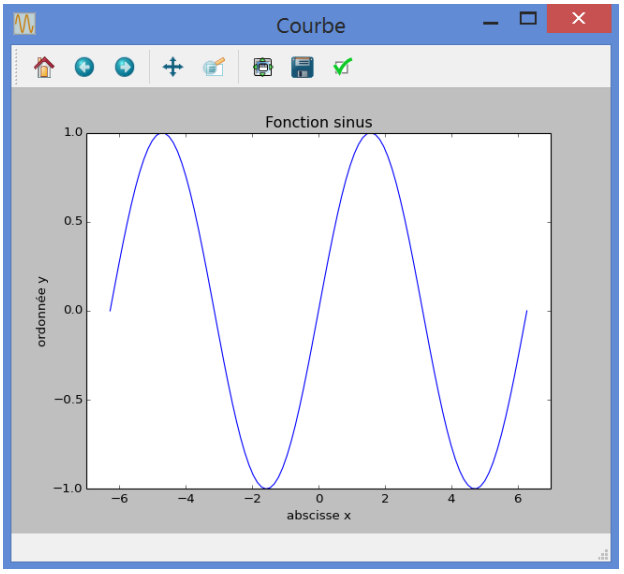


III. Graphiques

1. Courbes

Script Python (Courbes).

```
abs=np.linspace(-2*np.pi,2*np.pi,100)
ord=np.sin(abs)
pl.figure('Représentation de courbe')
pl.title('Fonction sinus')
pl.xlabel('abscisse x')
pl.ylabel('ordonnée y')
pl.xlim(-7,7)
pl.ylim(-1,1)
pl.plot(abs,ord)
pl.show()
pl.close()
```

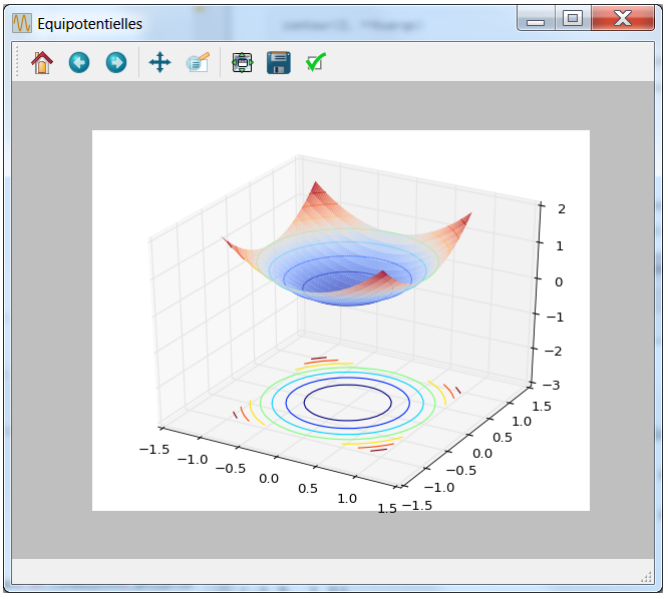


2. Lignes de niveaux

Définition (Lignes de niveaux).

Soit f une fonction de $\mathbb{R}^2$ dans $\mathbb{R}$ et k un réel.
On appelle **ligne de niveau** k **de** f l'ensemble des couples $(x, y) \in \mathbb{R}^2$ vérifiant $f(x, y) = k$.

Remarque : la ligne de niveau k de f peut être considérée comme l'intersection de la surface d'équation $z = f(x, y)$ et le plan d'équation $z = k$.



Pour représenter les lignes de niveaux, on utilise la fonction contour avec un maillage [2] fourni par la fonction meshgrid.

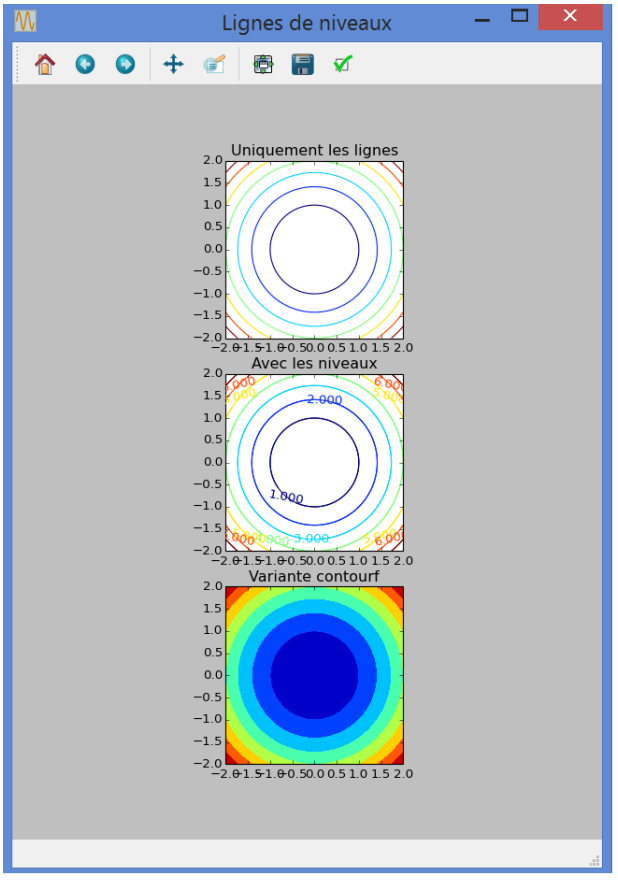
[2]. Discrétisation de l'espace de départ de f

Exemple Python (Maillage).

```
>>> x=[1,2,3]
>>> y=[10,20]
>>> m=pl.meshgrid(x,y)
>>> m
[array([[1, 2, 3],
[1, 2, 3]]), array([[10, 10, 10],
[20, 20, 20]])]
>>> X,Y=m
>>> X
array([[1, 2, 3],
[1, 2, 3]])
>>> Y
array([[10, 10, 10],
[20, 20, 20]])
```

Script Python (Lignes de niveaux).

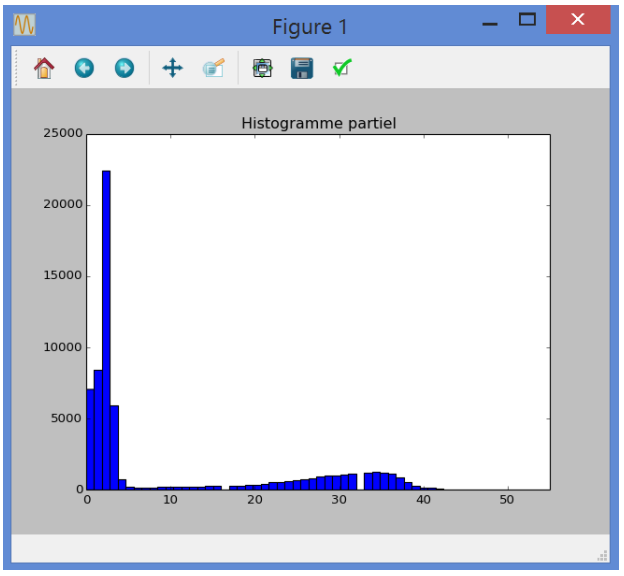
```
# Maillage
x=np.linspace(-2,2,100)
y=np.linspace(-2,2,100)
X,Y=np.meshgrid(x,y)
# Définition de la fonction
Z=X**2+Y**2
# Uniquement les lignes
pl.figure('Lignes de niveaux')
pl.subplot(1,3,1,aspect='equal')
pl.title('Uniquement les lignes')
pl.contour(X,Y,Z)
# Avec les niveaux
pl.subplot(1,3,2,aspect='equal')
pl.title('Avec les niveaux')
pl.contour(X,Y,Z)
ln=pl.contour(X,Y,Z)
pl.clabel(ln)
# Variante contourf
pl.subplot(1,3,3,aspect='equal')
pl.title('Variante contourf')
pl.contourf(X,Y,Z)
# Affichage
pl.show()
pl.close()
```



3. Histogrammes

Script Python (Histogrammes).

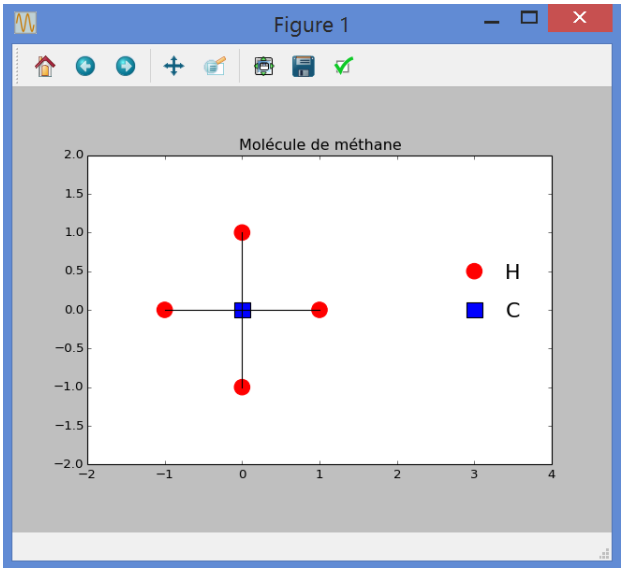
```
os.chdir('C:\\Users\\aridard2\\Desktop\\...
img=io.imread('coupeCerveau.jpg')
img.shape
img=img[:,:,0] # conservation d'un seul canal
data= img.reshape((len(img)**2,))
h=pl.hist(data,bins=256)
pl.title('Histogramme partiel')
pl.xlim(0,55)
pl.show()
pl.close()
```



4. Primitives graphiques

Script Python (Primitives graphiques).

```
H=[(-1,0),(1,0),(0,-1),(0,1)]
image=pl.subplot(1,1,1,aspect='equal')
pl.title ('Molécule de méthane')
for coord in H:
pl.plot([0,coord[0]],[0,coord[1]],color='black')
cercle=pl.Circle(coord,radius=0.1,color='r')
image.add_patch(cercle)
rect=pl.Rectangle((-0.1,-0.1),0.2,0.2)
image.add_patch(rect)
cercle2=pl.Circle((3,0.5),radius=0.1,color='r')
image.add_patch(cercle2)
pl.text(3.5,0.4,'H',color='black',fontsize=20)
rect2=pl.Rectangle((2.9,-0.1),0.2,0.2)
image.add_patch(rect2)
pl.text(3.5,-0.1,'C',color='black',fontsize=20)
pl.xlim(-2,4)
pl.ylim(-2,2)
pl.show()
pl.close()
```



Période 4

Bases De Données

I. Introduction

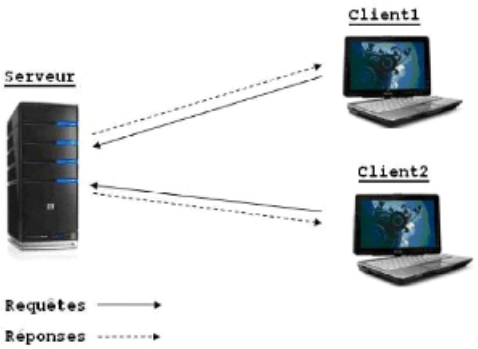
Considérons la feuille (de calcul) EXCEL suivante (extrait de données d'un établissement scolaire) :

	A	B	C	D	E	F	G	H	I	J	K	L
1	Id	Nom + Prénom	Sexe	ddn	VilleN	CP	Classe	Niveau	Spé	Langues	Fonction	DécompteFrais
2	1	BRAZAN Pierre	M	21/01/1992	Lille	59000	1A	HEI1	MP	ANG, ALL	délégué 1A	Scolarite 1230 ; livres 125
3	2	LE FRANCOIS Jacques	M	20/04/1990	Paris	75000	1A	HEI1	MP	ANG, ALL		Scolarite 1230 ; livres 185
4	3	VAN PERSIL Claire	F	01/05/1993	Lille	59000	1A	HEI1	MP	ANG, ALL	délégué HEI1 MP	Scolarite 1230 ; livres 125 ; photocopies 175
5	4	HADAMART Georges	M	12/12/1992	Marseille	13000	1A	HEI1	MP	ESP	délégué HEI1	Scolarite 1230 ; livres 210 ; photocopies 110
6	5	DE SOUTTER Jeanne	F	15/11/1992	Marseille	13000	1B	HEI1	MP	ANG, ALL		Scolarite 1290 ; livres 125
7	6	HAVEZ Jean-Pierre	M	09/01/1993	Lyon	69000	1B	HEI1	PC	ANG, ALL	délégué HEI1 PC	Scolarite 1290 ; livres 185
8	7	TORTUE Jacques	M	28/12/1992	Roubaix	59100	1B	HEI1	PC	ESP, ALL		Scolarite 1290 ; livres 125
9	8	BIZANCE France	F	15/08/1991	Tourcoing	59200	1B	HEI1	PC	ESP	délégué 1B	Scolarite 1290 ; livres 250
10	9	CARTIER Bruno	M	10/04/1992	Lille	59000	1B	HEI1	PC	ESP, ALL		Scolarite 1290 ; livres 125
11	10	ELIMIER François	M	22/05/1992	Roubaix	59100	1B	HEI1	PC	ANG, ESP		Scolarite 1290 ; livres 185
12	11	BRAZAN Paul	M	01/05/1990	Lille	59000	1B	HEI1	MP	ALL		Scolarite 1230 ; livres 125 ; sports 100

Il est assez facile, par tri et sélection, d'obtenir les élèves d'une classe donnée suivant la spécialité MP ou PC. Mais comment faire [1] pour obtenir les élèves de 1B suivant l'option langue espagnol et dont le décompte total des frais est supérieur à 1500 € ?
Par ailleurs, certaines données peuvent être redondantes (Lille et 59000 par exemple) et donc occuper de l'espace mémoire inutilement ce qui peut s'avérer problématique pour des données en très grand nombre. Voir, par exemple, le fichier (pwt80.xlsx) de la PennWorldTable (disponible librement sur Internet) qui regroupe des données économiques mondiales.
Aussi, une organisation des données sous forme d'un ensemble cohérent d'informations liées par des relations est utile. Ce type d'organisation est dit **base de données**.

1. Système de gestion de base de données : SGBD

C'est un logiciel contenant des données (sous forme de bases) et des fonctions manipulant ces données : définition, modification (extraction et « modification ») et contrôle (cohérence et protection) des données.



L'architecture d'un tel ensemble est souvent organisée sous la forme : **clients-serveur**. Cela signifie que les données et les fonctions sont gérés par le serveur (SQLite) et que des clients (interface SQLiteStudio ou Python) interrogent (requêtes) le serveur dans un langage (SQL) afin d'obtenir de celui-ci des réponses.
Il existe différents modèles de bases de données (objets, relationnels…) mais nous nous limiterons aux **bases relationnelles** qui s'appuient sur la théorie des ensembles et la notion mathématique de relations.

[1]. On aura la réponse à la fin de ce cours

2. Un peu de théorie : algèbre relationnelle

Assertions

Une assertion est une « phrase » booléenne : elle est soit vraie, soit fausse.

Exemples :
- 7 est pair ; 3 est un élément de $\{1,2,3,4,5\}$ sont des assertions
- Le nombre π a autant de chiffres pairs qu'impairs n'est pas une assertion

Prédicats

Une « phrase » dépendant de un ou plusieurs paramètres dont le remplacement par des valeurs (d'un ensemble donné) devient une assertion.

Exemples :
- $(x > 3)$ est un prédicat car si l'on remplace x par un entier, cela devient une assertion
- (x et y) sont des villes du Nord est un prédicat car si l'on remplace x et y par des villes françaises, cela devient une assertion
- $(\forall x \in \mathbb{R},\ x^2 \geq 0)$ n'est pas un prédicat, c'est une assertion !

Relations

Etant donnés des ensembles E_1, E_2, …, E_n, on forme l'ensemble produit $P = \prod_{i=1}^{n} E_i$.
Une relation R sur P est un sous-ensemble de P ; il est donc constitué de n-uplets.
Lorsque R est fini (et surtout pas trop grand), on peut le représenter de manière explicite sous forme de tableau.

Exemples :

Relation 1

Id	Nom + Prénom	Sexe	ddn	VilleN
1	BRAZAN Pierre	M	21/01/1992	Lille
2	LE FRANCOIS Jacques	M	20/04/1990	Paris
3	VAN PERSIL Claire	F	01/05/1993	Lille
4	HADAMART Georges	M	12/12/1992	Marseille
5	DE SOUTTER Jeanne	F	15/11/1992	Marseille
6	HAVEZ Jean-Pierre	M	09/01/1993	Lyon
7	TORTUE Jacques	M	28/12/1992	Roubaix
8	BIZANCE France	F	15/08/1991	Tourcoing
9	CARTIER Bruno	M	10/04/1992	Lille
10	ELIMIER François	M	22/05/1992	Roubaix
11	BRAZAN Paul	M	01/05/1990	Lille

Relation 2

Ville	CP
Lille	59000
Paris	75000
Marseille	13000
Lyon	69000
Roubaix	59100
Tourcoing	59200

Vocabulaire :
- **attribut**s d'une relation : noms des E_i
- **clé** d'une relation : tout attribut (ou toute famille d'attributs) permettant de distinguer les n-uplets
- **domaine** d'un attribut : ensemble contenant les valeurs de l'attribut

Exemples :
- Dans la relation 1, Id est une clé mais VilleN n'en est pas une
- Un domaine pour l'attribut Sexe est {« F », « M »}
- Dans la relation 2, Ville est une clé

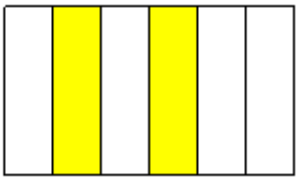
Les n-uplets d'une relation sont distincts deux à deux et il n'y a pas d'ordre sur ces n-uplets.

Opérations ensemblistes

Définition (Projection)**.**

Si R est une relation sur P et F un sous-produit de P, alors la **projection de** R **sur** F est la relation sur F définie par :

$$p_F(R) = \{(x_{j_1}, \ldots, x_{j_k}) \in F | x = (x_1, \ldots, x_n) \in R\} \quad \text{où } F = E_{j_1} \times \cdots \times E_{j_k}$$



Définition (Sélection)**.**

Si R est une relation sur P et Q un prédicat sur R, alors la **sélection de** R **suivant** Q est la relation sur R définie par :

$$\sigma_Q(R) = \{x \in R | x \text{ vérifie } Q\}$$

Définition (Produit)**.**

Si R_1 et R_2 sont des relations sur P_1 et P_2, alors le **produit de** R_1 **par** R_2 est la relation sur $P_1 \times P_2$ définie par :

$$R_1 \times R_2 = \{(x, y) \in P_1 \times P_2 | x \in R_1 \text{ et } y \in R_2\}$$

Définition (Jointure)**.**

Si R_1 et R_2 sont des relations sur P_1 et P_2, Q un prédicat sur $R_1 \times R_2$, alors la **jointure de** R_1 **et** R_2 **suivant** Q est la relation sur $R_1 \times R_2$ définie par :

$$j_Q(R_1, R_2) = \sigma_Q(R1 \times R2)$$

Exemples :

- Soit R_1, R_2 les relations suivantes et Q le prédicat : Att12 = Att21.

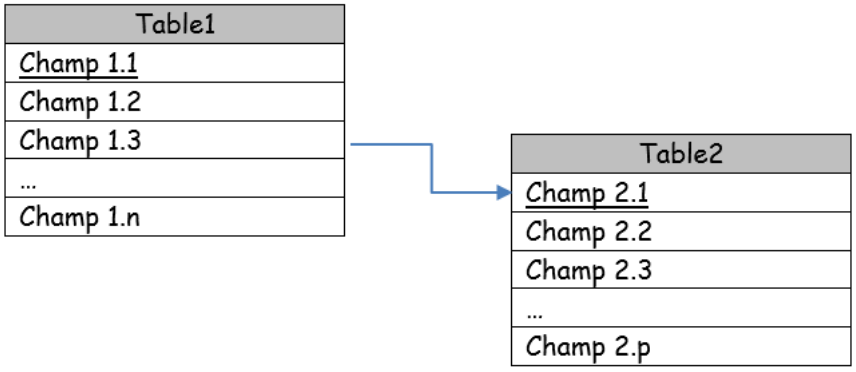
R1	
Att11	Att12
A1	B1
A2	B1
A2	B3

R2	
Att21	Att22
B1	D1
B2	D2

  Déterminer la jointure de R_1 et R_2 suivant Q, puis la projection sur Att11 × Att12 × Att22 de cette jointure.
- Que fournit la projection sur Id × Nom + Prénom × Sexe × ddn × VilleN × CP de la jointure de R_1 et R_2 suivant Q où R_1, R_2 sont les relations 1 et 2 de l'exemple de la page précédente, et Q le prédicat VilleN=Ville ?

II. Application aux bases de données

Une **table** est un ensemble (de données) semblable à une relation, mais acceptant dans certains cas (résultats de requêtes) des doublons parmi les n-uplets.
Les **enregistrements** d'une table sont les n-uplets et les **champs** d'une table sont les attributs.
Les champs d'une table sont typés : entiers, flottants, chaîne de caractères, ... On peut éventuellement limiter ces types : entiers compris entre 0 et 999 ... On parlera alors du **domaine** du champ.
Souvent, une **clé** dite **primair**e est choisie. **Une clé étrangère** est un champ qui est clé primaire d'une autre table.
Afin d'extraire des données d'une base, l'algèbre relationnelle est appliquée aux tables de la base.
Chaque table peut se schématiser par un tableau :



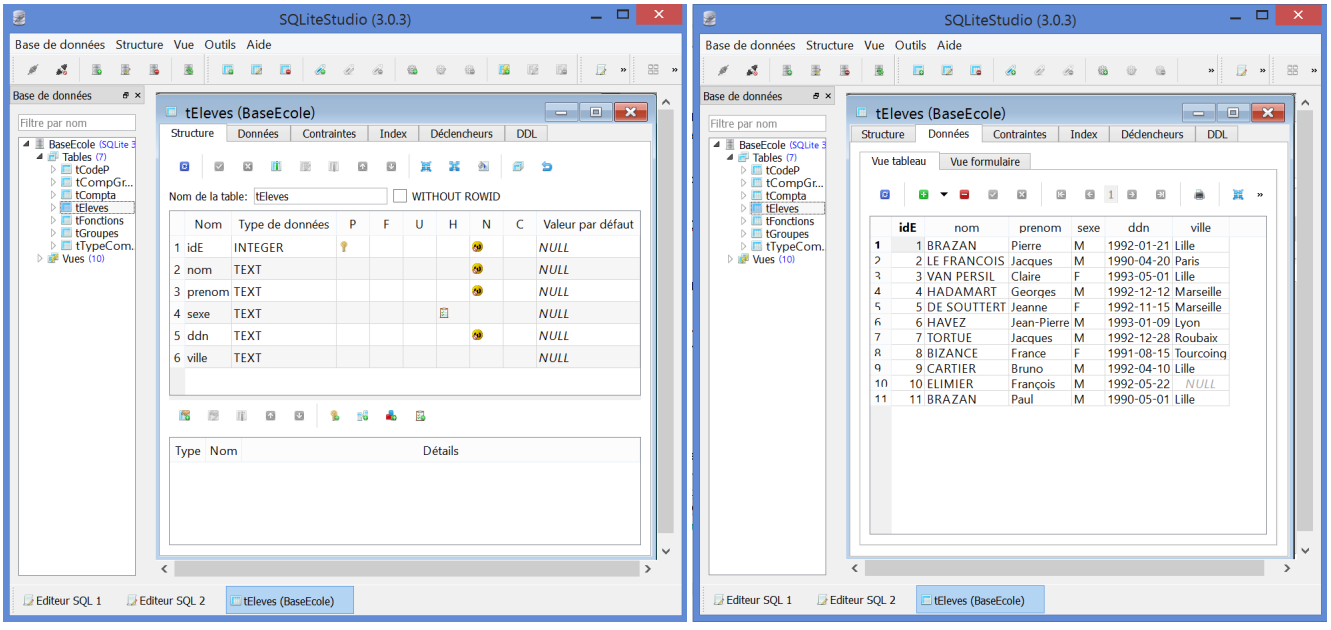
Le soulignement permet de visualiser les clés primaires.
Quant aux clés secondaires, elles sont indiquées par des flèches (Champ1.3 dans la Table 1)

III. Langage des SGBD : SQL

Chaque SGBD dispose d'un langage, pour nous ce sera le SQL (Structured Query Language). Ce langage (comme les autres) se décompose en trois types suivant son utilisation :
- LDD (Langage de Définition des Données)
- LMD (Langage de Modification des Données)
- LCD (Langage de Contrôle des Données)

La casse des caractères n'est pas significative en SQL (il ne fait pas la différence entre majuscule et minuscule).

Ce cours sera illustré à partir de la base appelée *BaseEcole*. En particulier, on utilisera la table *tEleves* comportant 6 champs. La clé primaire étant le champ *idE* de type entier.



IV. Premières requêtes en SQL

1. Création d'une table virtuelle

Pour créer une table **virtuelle** identique à la table T, on exécute la commande suivante :

Interrogation (création d'une table virtuelle)**.**

select * from T

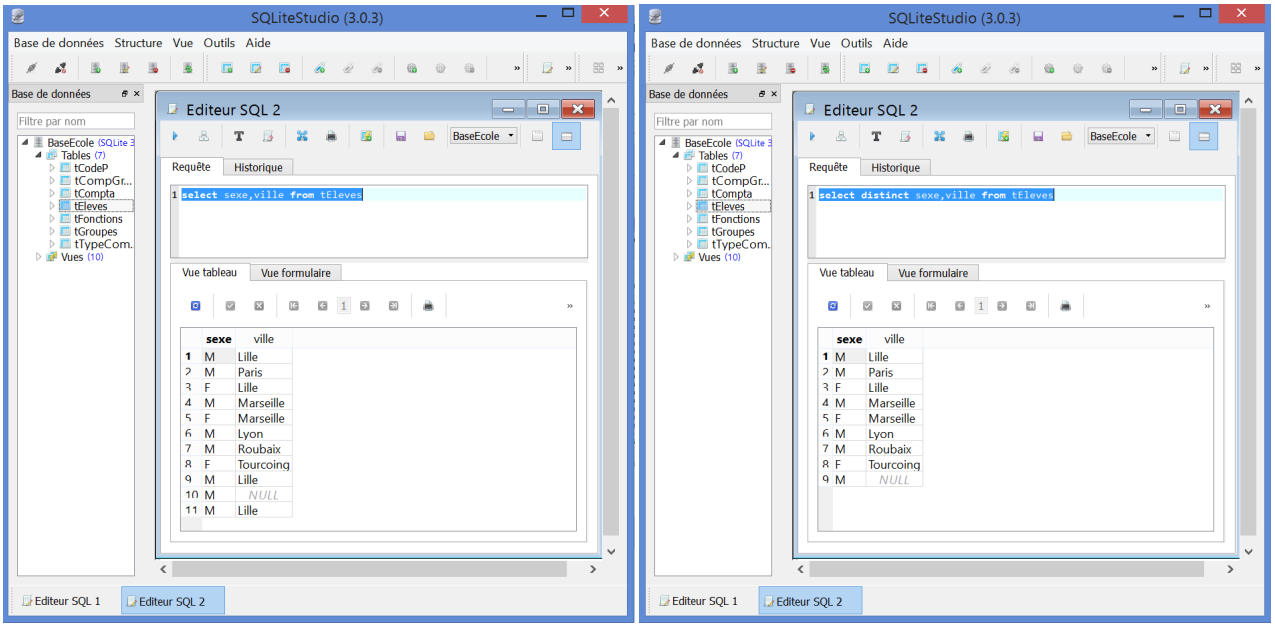
Contrairement aux tables initiales, les tables virtuelles créées peuvent être conservées (sous forme de vues [2]) ou non.

On peut aussi préciser la création en exploitant les opérations ensemblistes.

2. Projection

Interrogation (projection)**.**

select [distinct] Ch1,Ch2, . . . from T * pour tous les champs



Remarque : pour éviter les doublons (possibles si aucune clé primaire n'est choisie), on utilise **distinct**.

3. Sélection

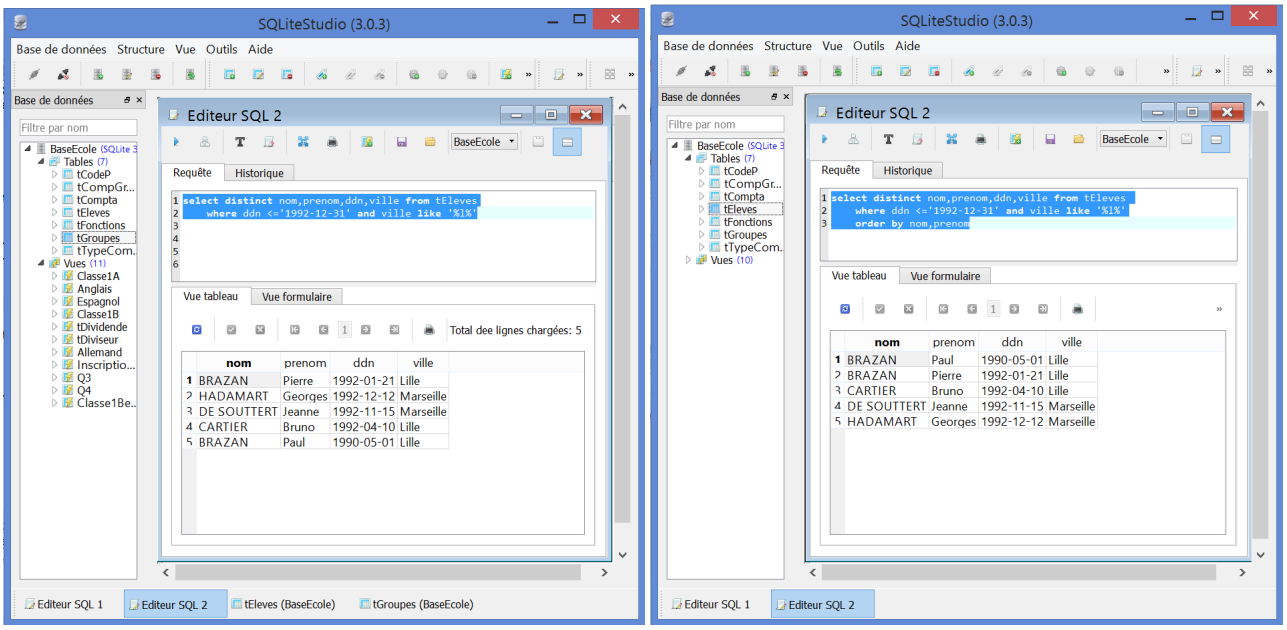
Interrogation (sélection)**.**

select . . . from T where P

où P désigne un prédicat défini à l'aide d'un :

- opérateur logique : and , or , not
- opérateur de comparaison : < , > , <= , >= , =
- opérateur d'appartenance : in syntaxe : in(val1,. . .,valp)
- filtre : like
  - le caractère _ remplace un caractère
  - le caractère % remplace une chaîne de caractères
- test d'indétermination : is null , is not null

[2]. A l'aide de la fonction **create** ou directement avec le bouton



Remarque : pour trier la table virtuelle, on utilise **order by**.

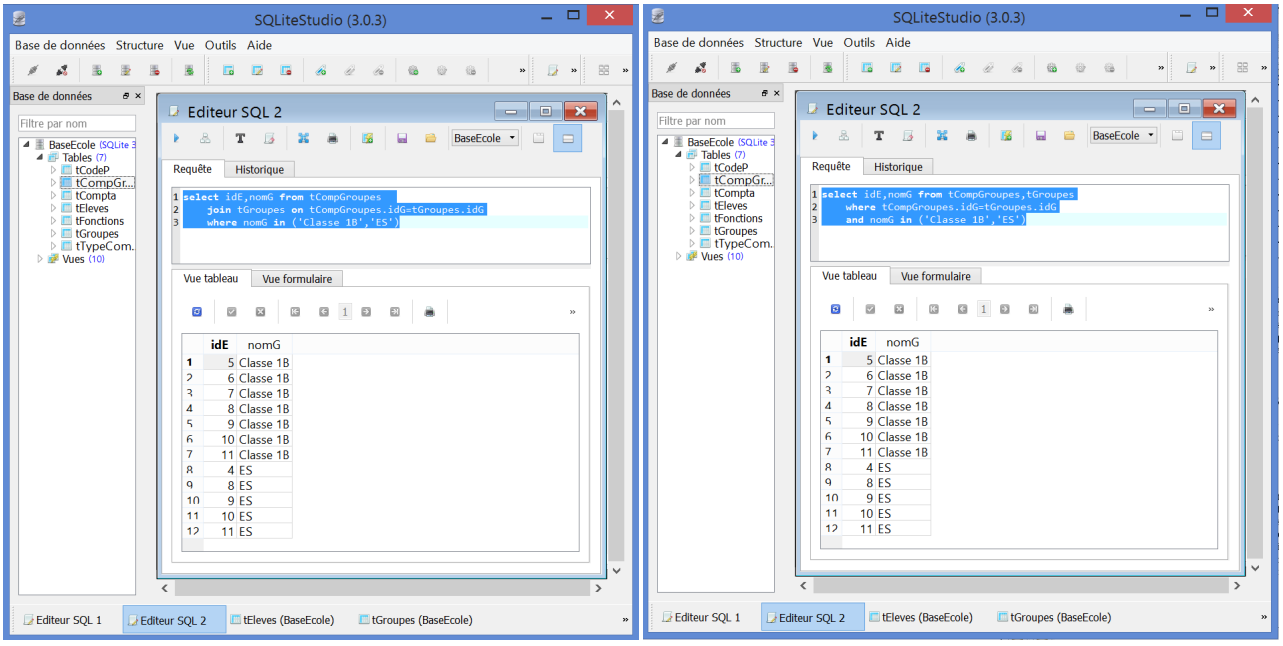
V. Jointures

Interrogation (jointure)**.**

select . . . from T1 join T2 on P

ou encore

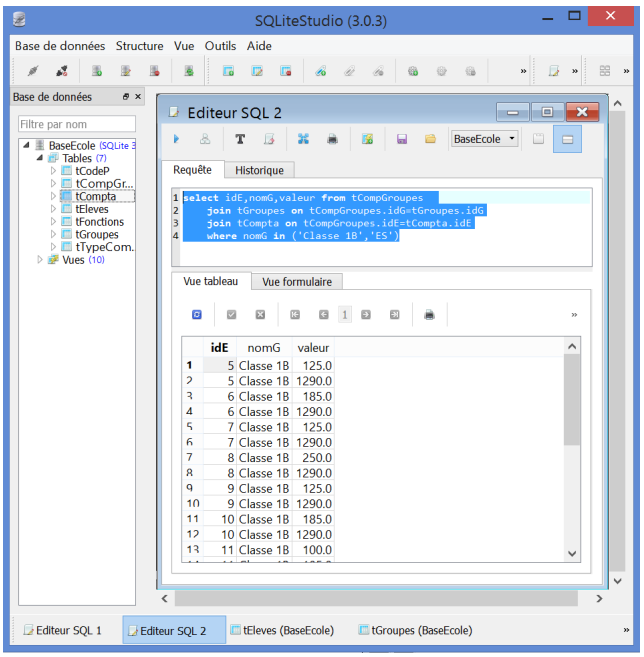
select . . . from T1,T2 where P

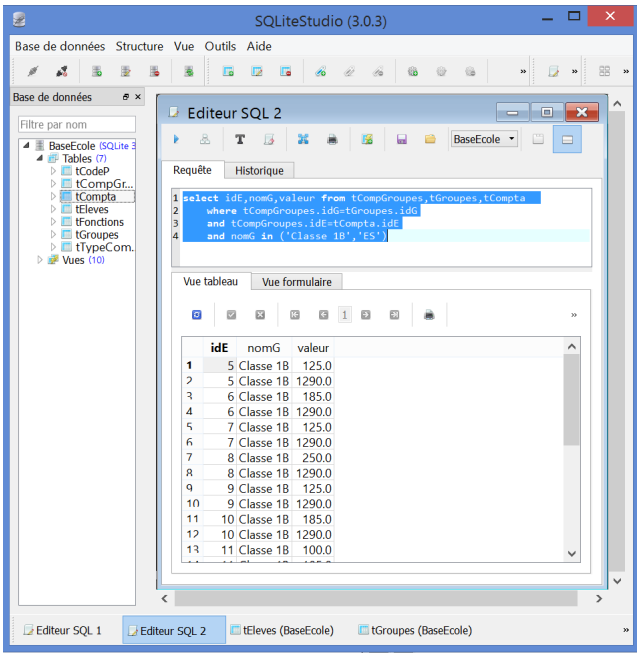


Remarques :

- Dans l'exemple précédent, la jointure est suivie d'une sélection : nomG in ('Classe1B','ES')
- Lorsque deux tables ont des champs de même nom, il faut les distinguer en préfixant le nom du champ par celui de la table
- La deuxième syntaxe met en évidence la jointure comme un produit suivie d'une sélection : tCompGroupes.idG=tGroupes.idG
- La première syntaxe permet de différencier le prédicat de jointure (on ...) de l'autre (where ...)

On peut également faire des jointures multiples :





VI. Fonctions d'agrégation et partitions

Interrogation (agrégat).

select . . . , fct(Ch) , . . . from T . . .

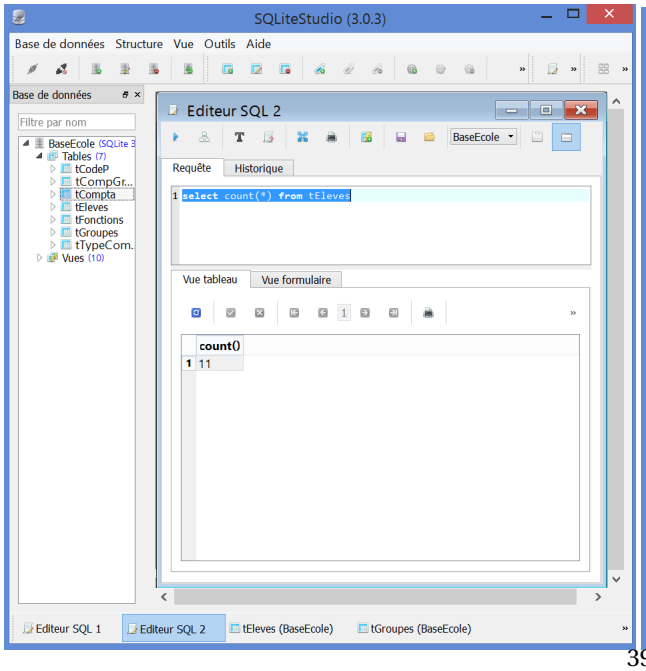
où fct désigne une des fonctions d'agrégation suivantes :

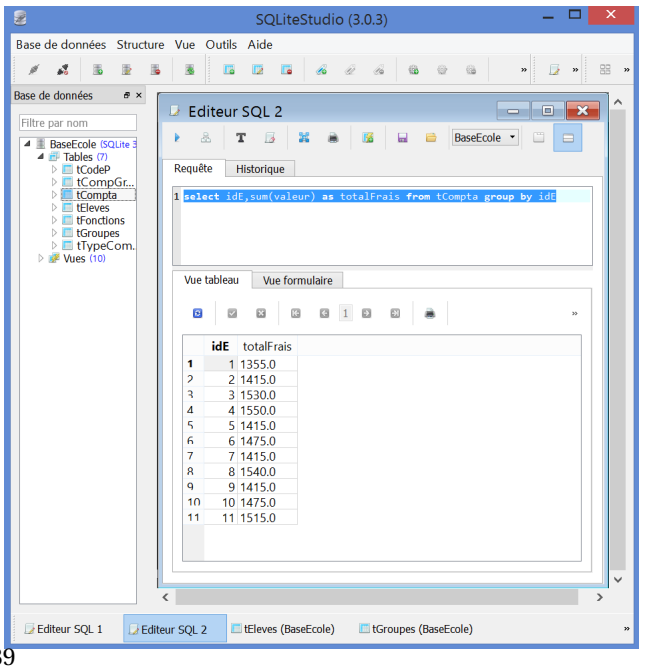
- sum
- max
- min
- avg
- count

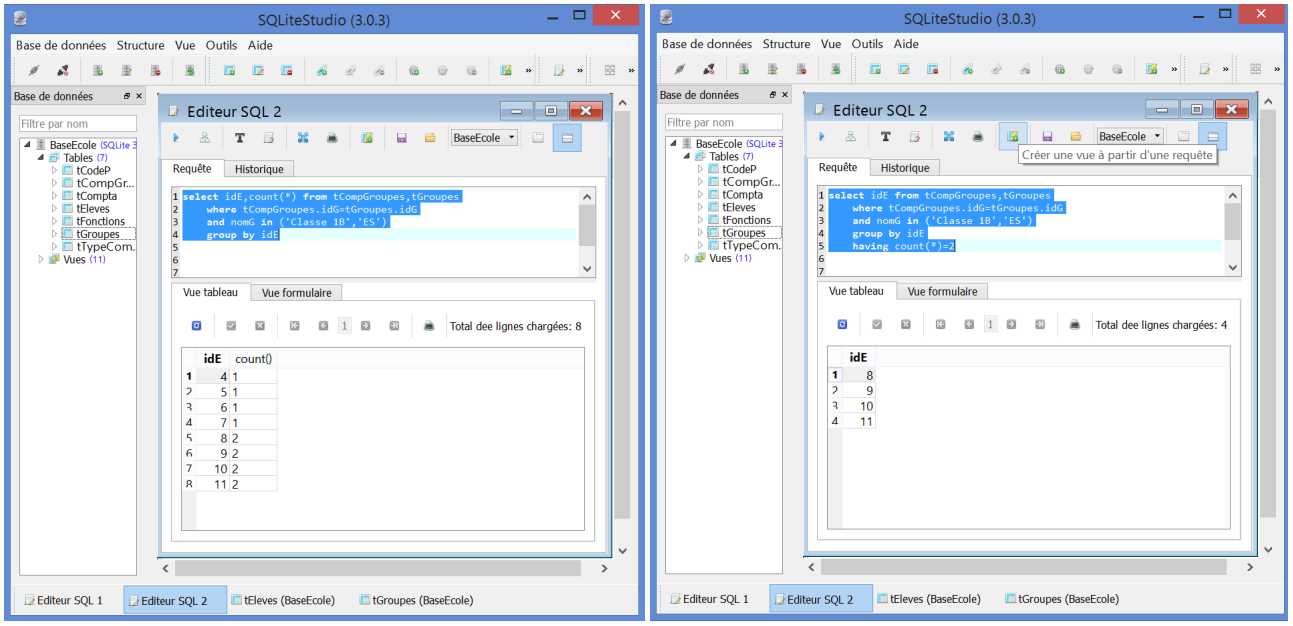
Interrogation (partition).

select . . . , fct(Ch) , . . . from T . . . group by Ch [having Q]

où Q désigne un prédicat portant sur le champ Ch





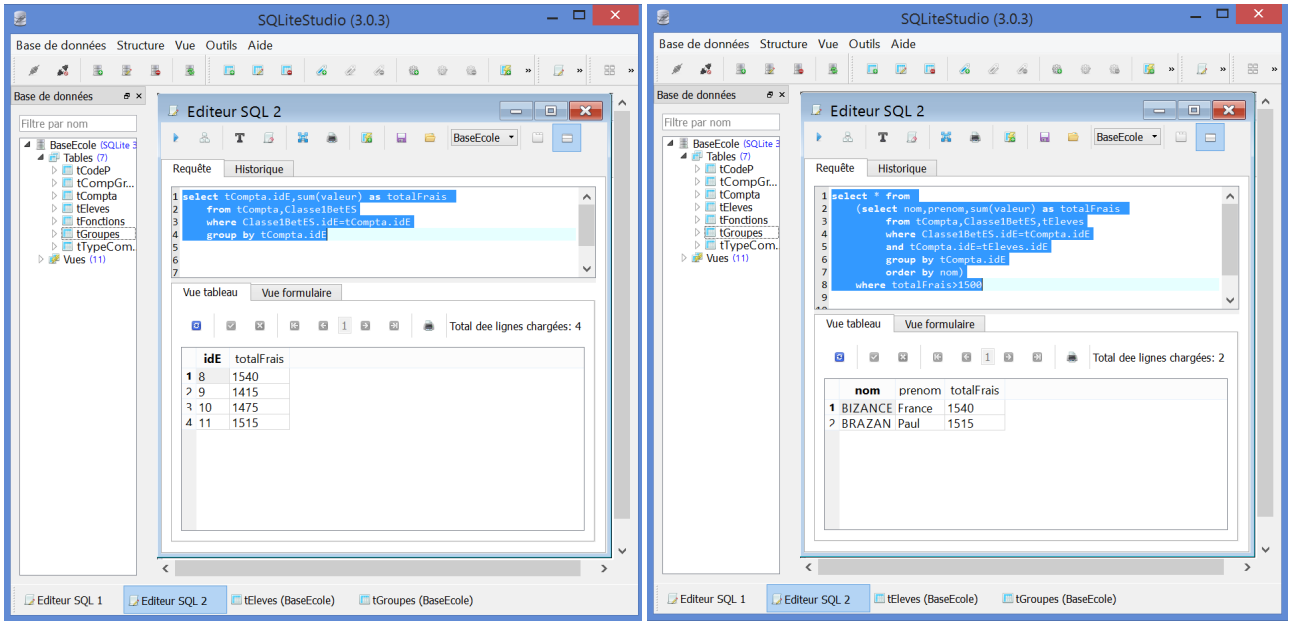


Remarque : cette dernière table virtuelle a été enregistrée sous le nom de *Classe1BetES*

Interrogation (syntaxe générale).

select … from … where … group by … having … order by …

VII. Utilisation des vues et des sous-requêtes



Annexes

Aide mémoire Python

Python (Types de base).

int , float , bool — entier , flottant , booléen

Python (Opérateurs numériques).

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

Python (Conteneurs).

Non modifiables

str	'hei' ou "hei"	chaines de caractères
tuple	(421,3.14,'hei')	t-uplets

Modifiables

list	[421,3.14,'hei']	listes
dict	{clé : valeur}	dictionnaires **(MP)**

Python (Opérations sur les conteneurs).

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

Opérations spécifiques aux listes

lst.insert(idx,val) — insertion de val dans lst à la position idx
del(lst[idx]) — suppression dans lst de l'élément d'index idx

Python (Chaines de caractères).

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

"−".join(['Jean','Pierre']) — donne 'Jean−Pierre'
("−" : chaine de jointure ; ['Jean','Pierre'] : séquence de chaines)

"2 ;5.1 ;12".split(";") — donne ['2','5.1','12']
(";" : chaine de séparation)

Python (Indexation des séquences).

Pour les listes, tuples, chaines de caractères, ...

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

Python (Affichage/Saisie).

print(ch,"contient",len(ch),"caractères") — affichage
e=input("message facultatif") — affectation à e de la chaine de caractères saisie

Python (Déclaration de fonction).

```
def nomfct (a,b) :
    bloc d'instructions
    return(res)
```

avec a et b des paramètres éventuels

Python (Fichiers).

f=open('chemin\ nomfichier.txt' , 'r' / 'w' / 'a') — ouverture en lecture / écriture / ajout

f.readline() — lecture d'une ligne (caractère de fin de ligne compris) à partir de l'endroit où l'on se trouve dans le fichier

f.readlines() — séquence des lignes du fichier

f.write('texte à écrire') — écriture à partir de l'endroit où l'on se trouve dans le fichier

f.close() — fermeture

Python (Instructions conditionnelles).

```
if b1 :
    |bloc d'instructions si b1 est vrai
elif b2 :
    |bloc d'instructions si b1 est faux et b2 vrai
else :
    |bloc d'instructions sinon
```

Plusieurs elif possibles

Python (Boucle itérative).

for *i* in range([*a*,] *b* [,*p*]) **:** — i varie de a à b exclu avec un pas de p
(0 par défaut ; 1 par défaut ; *b* : exclu)

```
    |bloc d'instructions
```

Extension du for aux conteneurs

for *i* in cont **:** — i prend ses valeurs dans cont

```
    |bloc d'instructions
```

Python (Boucle conditionnelle).

while *b* **:** — instructions éxécutées tant que b est vrai

```
    |bloc d'instructions
```

Compléments sur les modules en Python

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

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(nbElts 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

Module (**copy** : gestion des « copies »).

from copy import deepcopy	importation de la fonction deepcopy
deepcopy(var)	Création d'une copie « indépendante »

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

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

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

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

COMMANDES SQL

Interrogation (syntaxe générale)**.**

select ... from ... where ... group by ... having ... order by ...

Interrogation (projection)**.**

select [distinct] Ch1, ... * pour tous les champs

Interrogation (sélection)**.**

... where P

où P désigne un prédicat défini à l'aide d'un :
- opérateur logique : and , or , not
- opérateur de comparaison : < , > , <= , >= , =
- opérateur d'appartenance : in syntaxe : in(val1,...,valp)
- filtre : like
  - le caractère _ remplace un caractère
  - le caractère % remplace une chaîne de caractères
- test d'indétermination : is null , is not null

Interrogation (union, intersection, différence)**.**

select * from T1[where P1]
union
select * from T2 [where P2]

select * from T1[where P1]
intersect
select * from T2 [where P2]

select * from T1[where P1]
except
select * from T2 [where P2]

Interrogation (jointure)**.**

... from T1 join T2 on P

ou encore

... from T1,T2 where P

Interrogation (agrégat)**.**

select ... , fct(Ch) , ...

où fct désigne une des fonctions d'agrégation suivantes :
- sum
- max
- min
- avg
- count

Interrogation (partition)**.**

... group by Ch [having Q]

où Q désigne un prédicat portant sur le champ Ch

Interrogation (tri)**.**

... order by [desc] Ch1, ...

Interrogation (alias)**.**

select ... , Ch as nomChamp , ... from T as nomTable

Interrogation (sous-requête)**.**

... from (select ...)

Modification (insertion)**.**

insert into T values (valCh1 , ... , valChn)

Modification (mise à jour)**.**

update T set Chk=newValChk , ... [where P]

Modification (suppression)**.**

delete from T [where P]