# Introduction à Python

V. Monbet
(document fortement inspiré du livre de R. Cordeau
et des supports de cours de J.-P. Becirspahic)


UFR de Mathématiques
Université de Rennes 1

15 octobre 2018

# Outline

# Outline

# Outline

# Numpy, premiers pas

- NUMPY est une librairie pour les tableaux multidimensionnels, en particulier les matrices ;
- Son implémentation est proche du hardware, et donc beaucoup plus efficace pour les calculs que si on code les calculs via des boucles ;

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6]])
A
Out[2]:
array([[1, 2, 3],
       [4, 5, 6]])
A.ndim
Out[3]: 2
A.shape
Out[4]: (2, 3)
A.dtype
Out[5]: dtype('int64')
A = np.array([[1, 2, .3], [4, 5, 6]])
A.dtype
Out[7]: dtype('float64')
```

# Changer le type d'un tableau

- On peut changer le type des éléments d'un tableau de float vers int et de int vers float

```
A = np.array([[1, 2, .3], [4, 5, 6]])
A.dtype
Out[7]: dtype('float64')
X = np.asfarray(A)
Out[8]: X.dtype
dtype('float64')
Out[9]: X
array([[ 1. ,  2. ,  0.3],
       [ 4. ,  5. ,  6. ]])
Out[10]: np.int_(X)
array([[1, 2, 0],
       [4, 5, 6]])
```

# Numpy, créer des tableaux

- En pratique, on crée rarement des tableaux à la main.
- Il existe par exemple des outils pour créer des tableaux de suites incrémentées

## Suites

```
import numpy as np
debut = 0.5
fin = 3.5
pas = 1
A = np.arange(debut,fin,pas)
A
Out[24]: array([ 0.5,  1.5,  2.5])
B = np.linspace(debut,fin,nb_points)
B
Out[28]: array([ 0.5 ,  1.25,  2.  ,  2.75,  3.5 ])
```

# numpy, créer des tableaux

- pour des tableaux de constantes

## Tableaux constants

```
import numpy as np

A = np.zeros(5)
A
Out[29]: array([ 0.,  0.,  0.,  0.,  0.])
B = np.ones((3,4))
print(B)
Out[30]:
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
C = np.ones((2,3,2)) # tableau a 3 dimensions
print(C)
Out[37]:
array([[[ 1.,  1.],
        [ 1.,  1.],
        [ 1.,  1.]],

       [[ 1.,  1.],
        [ 1.,  1.],
        [ 1.,  1.]]])
```

# Numpy, créer des tableaux

- En pratique, on crée rarement des tableaux à la main.
- pour des matrices diagonales

## Matrices diagonales

```
import numpy as np
>>> I = np.eye(3)
>>> I
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> D = np.diag([3,2,4,1])
>>> D
array([[3, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 4, 0],
       [0, 0, 0, 1]])
```

# Exercices

1. Écrire un programme PYTHON pour créer une matrice 3x3 contenant les valeurs 2 à 10.
2. Écrire un programme PYTHON pour créer un vecteur nul de taille 10 et puis mettre à jour la sixième valeur à 11.
3. Écrire un programme PYTHON pour convertir un tableau en type flottant.
   Tableau original : [1, 2, 3, 4]
   Tableau converti en type flottant : [1., 2., 3., 4.]

# Numpy, slicing

- On peut accéder aux éléments des tableaux

```
>>> A = np.array([[1, 2, 3,4],
        [5, 6,7,8]])
>>> A
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> A[0,1]
2
```

# Numpy, slicing

- On peut accéder à des éléments des tableaux par lignes ou colonnes

```
>>> A = np.array([[1, 2, 3,4],
       [5, 6,7,8]])
>>> A[0,:] # premiere ligne
array([1, 2, 3, 4])
>>> A[:,3] # quatrieme colonne
array([4, 8])
>>> A[:,0:2]
array([[1, 2],
       [5, 6]])
```

# Numpy, méthode take

Extraction de sous matrices si *a* est unidimensionnel,

```
>>> a = [4, 3, 5, 7, 6, 8]
>>> indices = [0, 1, 4]
>>> np.take(a, indices)
array([4, 3, 6])
```

ou

```
>>> A = np.array(a)
>>> A[indices]
array([4, 3, 6])
```

Si `indices` est un tableau, la sortie est un tableau de même dimension

```
np.take(A, [[0, 1], [2, 3]])
array([[4, 3],
       [5, 7]])
```

# Numpy, méthode take

Extraction de sous matrices si $A$ est multidimensionnel,

```
 >>> A = np.array([4, 3, 5, 7, 6, 8,9,10,11]).reshape(3,3) # matrice 3
 >>> A.take([0,1],0)
array([[4, 3, 5],
       [7, 6, 8]])
>>> A.take([0,2],1)
array([[ 4,  5],
       [ 7,  8],
       [ 9, 11]])
>>> A[:2,:2]
array([[4, 3],
       [7, 6]])
```

# Numpy, slicing

- On peut modifier les valeurs d'un tableau par élément ou sous tableaux

```
>>> A[0,2:] = 0
>>> A
array([[1, 2, 0, 0],
       [5, 6, 7, 8]])
>>> B = np.arange(4,0,-1).reshape((2,2))
>>> B
array([[4, 3],
       [2, 1]])
>>> A[:2,:2] = B
>>> A
array([[4, 3, 0, 0],
       [2, 1, 7, 8]])
```

- On note ci-dessus le `reshape` qui permet de modifier la taille d'un tableau.

# copy or not copy

```
>>> T1 = np.zeros((2, 3),
     dtype="uint8")
array([[0, 0, 0],
       [0, 0, 0]], dtype=uint8)
>>> T1[0, 1] = 128
>>> T2 = T1[:, 1:]
>>> T2[1, 1] = 50
array([[128,   0],
       [  0,
50]], dtype=uint8)
>>> T1[0, 2] = 23
>>> T2
array([[128,  23],
       [  0,
50]], dtype=uint8)
```

```
>>> T3 = T2.copy()
>>> T3[1, 1] = 17
>>> T3
array([[128,  23],
       [  0,
17]], dtype=uint8)
>>> T2
array([[128,  23],
       [  0,
50]], dtype=uint8)
```

# NUMPY, opérations sur les tableaux

- Les opérations sur les tableaux peuvent se faire terme à terme. En particulier, les opérations simples comme `+, -, *, /, **` sont effectuées sur chaque élément du tableau.

## Opérations simples

```
>>> B
array([[4, 3],
       [2, 1]])
>>> B+1
array([[5, 4],
       [3, 2]])
>>> B**2
array([[16,  9],
       [ 4,  1]])
```

- NUMPY dispose aussi de fonctions scientifiques, par exemple : `np.exp()`, `np.cos()`, $\cdots$

# NUMPY, opérations sur les tableaux

- NUMPY permet bien sûr de faire des opérations matricielles comme la transposition, le calcul d'une trace

```
>>> B
array([[4, 3],
       [2, 1]])
>>> B.T      # transposition
array([[4, 2],
       [3, 1]])
>>> B.trace()    # trace
5
```

# NUMPY, opérations sur les tableaux

- Produit matriciel

```
>>> B
array([[4, 3],
       [2, 1]])
>>> np.dot(B,B)  # calcule B*B (produit matriciel)
array([[22, 15],
       [10,  7]])
```

- Produit matrice-vecteur

```
>>> M = np.arange(16).reshape(4,4)
>>> u = [1,2,3,4]
>>> np.dot(B,u)  # calcule B*u (produit matriciel)
```

# Opérations sur les tableaux

- Opérations sur l'ensemble du tableau comme trouver la valeur minimum ou maximum, calculer la somme ou le produit de tous les éléments du tableau, ...

```
>>> B
array([[4, 3],
       [2, 1]])
>>> np.amin(B)
1
>>> np.sum(B)
10
```

# Trouver des éléments du tableau : np.nonzero

- Il est souvent utile de retrouver les éléments d'un tableau qui vérifient certaines conditions

```
>>> a
array([[2, 3, 4],
[4, 5, 6]])
 >>> i = np.nonzero(a > 3)
>>>  i
(array([0, 1, 1, 1]), array([2, 0, 1, 2]))
>>>  i[0]
array([0, 1, 1, 1])
>>> a[i]
array([4, 4, 5, 6])
>>> a[i] = 0
>>> a
array([[2, 3, 0],
[0, 0, 0]])
```

# Trouver des éléments du tableau : np.where

```
>>> Y = np.arange(16).reshape(4,4)
>>> Y
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])

 >>> np.where(Y \% 2 ==0)    # attention lire modulo...
(array([0, 0, 1, 1, 2, 2, 3, 3]), array([0, 2, 0, 2, 0, 2, 0, 2]))
```

# Exercices

(1) Écrire un programme pour renverser un tableau.
Tableau : [12 13 14 15 16 17 18 19 20 ]
Tableau renversé : [ 20 19 18 17 16 15 14 13 12]

(2) Écrire un programme pour transformer un tableau de type **`int`** en un tableau de type **`float`**.

(3) Écrire un programme qui ajoute des 0 tout autour d'un tableau.

```
Original array          0 on the border
[[ 1. 1. 1.]            [[ 0. 0. 0. 0. 0.]
[ 1. 1. 1.]             [ 0. 1. 1. 1. 0.]
[ 1. 1. 1.]]            [ 0. 1. 3. 1. 0.]
                        [ 0. 1. 1. 1. 0.]
                        [ 0. 0. 0. 0. 0.]]
```

(4) Écrire un programme qui renvoie un tableau dont les lignes sont renversées.

```
Original array          with reversed lines
[[ 1. 2. 3. 4.]            [[ 4. 3. 2. 1.]
[ 5. 6. 7. 8.]             [ 8. 7. 6. 5.]
[ 9. 10. 11. 12.]]            [ 12. 11. 10. 9.] ]
```

(4) Écrire un programme extrait les indices des éléments supérieurs à 10.

```
Original array
[[ 0. 10. 20. 30.]
[ 1. 20. 30. 4.]]
```

Banque d'exercices : https ://www.w3resource.com/python-exercises/list/

# Exercices

## solutions

(1) Écrire un programme pour renverser un tableau.

```
import numpy as np
x = np.arange(12, 21)
print("Original array:")
print(x)
print("Reverse array:")
x = x[::-1]
print(x)
```

(2) Écrire un programme pour transformer un tableau de type **int** en un tableau de type **float**.

```
import numpy as np
a = [1, 2, 3, 4]
x = np.array(a,dtype=float)
print(x)
x = np.asarray(a,dtype=float)
```

# Exercices

## solutions

(3) Écrire un programme qui ajoute des 0 tout autour d'un tableau.

```
import numpy as np
x = np.ones((3,3))
print("Original array:")
print(x)
y = np.zeros((x.shape[0]+2,x.shape[1]+2))
y[1:-1,1:-1] = x
print(y)
```

Autre solution

```
x = np.pad(x, pad_width=1, mode='constant', constant_values=0)
```

# Exercices

## solutions

(3) Écrire un programme qui renvoie un tableau dont les lignes sont renversées.

```
import numpy as np
x = np.arange(12).reshape(3,4)
print("Original array:")
print(x)
y = x[:,range(np.shape[1],-1,-1)]
print(y)
```

(4) Écrire un programme qui les valeurs puis les indices des éléments supérieurs à 10.

```
import numpy as np
x = np.array([[0, 10, 20], [20, 30, 40]])
print("Original array: ")
print(x)
print("Values bigger than 10 =", x[x>10])
print("Their indices are ", np.nonzero(x > 10))
```

# Outline

# Outline

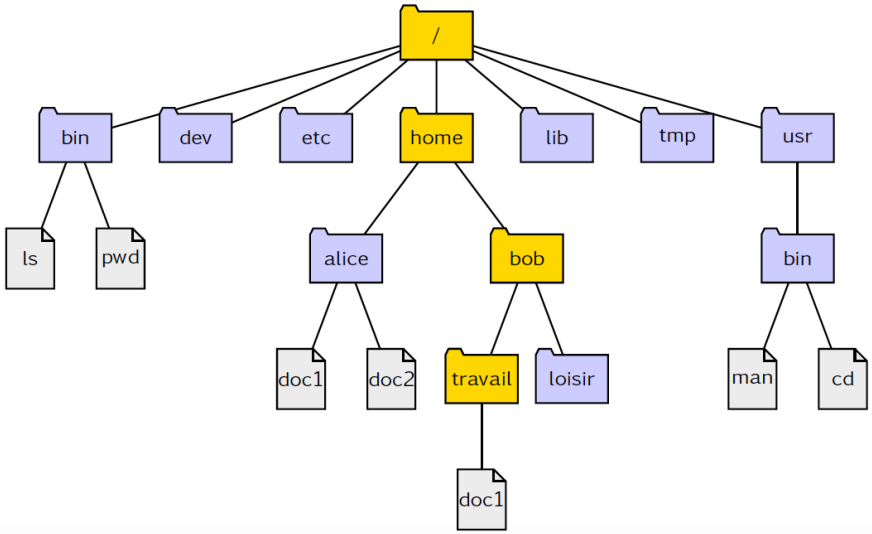
# Le module os

- Les instructions permettant à l"interprète de dialoguer avec le système d"exploitation font partie du module os :

```
import os
```

La fonction listdir liste le contenu d"un répertoire :

```
>>> os.listdir('/home/bob/travail')
['doc1']
```



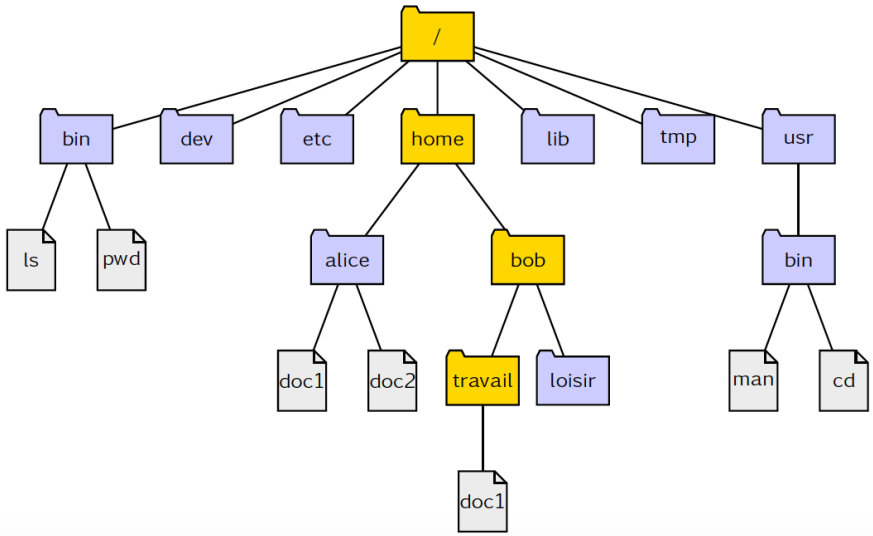
# Le module os

La fonction `getcwd` permet d'afficher le répertoire courant :

```
>>> os.getcwd()
```

La fonction `chdir` permet de changer de répertoire :

```
>>> os.chdir('home/alice')
```



# Outline

# Lecture d'un fichier texte

La fonction **open** propose trois modes d"ouverture d"un fichier :
- en lecture ('r') ;
- en écriture ('w') ;
- en ajout ('a').

Pour ouvrir en lecture le fichier exemple.txt du répertoire courant :

```
>>> comptine = open('exemple.txt', 'r')
```

Nous venons de créer un objet comptine faisant référence au fichier exemple.txt :

```
>>> comptine
<_io.TextIOWrapper name='exemple.txt' mode='r' encoding='UTF-8'>
```

Cet objet est un flux : les caractères sont lisibles uniquement les uns après les autres, sans possibilité de retour en arrière ni de saut en avant.

# Lecture d'un fichier texte

Pour lire le fichier dans son entier : la méthode read().

```
>>> comptine.open('r')
>>> comptine.read()
'Am, stram, gram,\nPic et pic et colegram,\nBour et bour et
ratatam,\nAm, stram, gram.'
>>> comptine.close()
```

Pour lire le fichier ligne par ligne : la méthode readlines(n).

```
>>> comptine.open('r')
>>> comptine.readlines()
['Am, stram, gram,\n', 'Pic et pic et colegram,\n',
'Bour et bour et ratatam,\n', 'Am, stram, gram.']
>>> comptine.close()
```

# Lecture d'un fichier texte

Lecture par énumération des lignes :

```
>>> comptine.open('r')
>>> n = 0
>>> for l in comptine:
... n += 1
... print('{} :'.format(n), l, end='')
1 : Am, stram, gram
2 : Pic et pic et colegram,
3 : Bour et bour et ratatam,
4 : Am, stram, gram.
>>> comptine.close()
```

# Lecture avec le module numpy

```
import numpy as np
from urllib.request import urlopen
url="https://perso.univ-rennes1.fr/valerie.monbet/PYTHON/
    LePlusRare_inputs.tex"

abscisses=np.loadtxt(urlopen(url),usecols=1,delimiter=",")
abscisses = list(abscisses)
ordonnees=np.loadtxt(urlopen(url),usecols=2,delimiter=",")
ordonnees = list(ordonnes)
noms=np.genfromtxt(urlopen(url),
     dtype='str',usecols=0,delimiter=",")
noms = list(noms)
```

```
import numpy as np
from urllib.request import urlopen
url="https://perso.univ-rennes1.fr/valerie.monbet/PYTHON/
    LePlusRare_inputs.tex"
coord=np.loadtxt(urlopen(url),usecols=range(1,3),delimiter=",")
```

# Outline

# Fichiers image

## Images en noir et blanc

Une image binaire peut être représentée par une matrice p×q dont les éléments, des 0 ou des 1 (plus exactement des booléens), indiquent la couleur du pixel : 0 pour le noir et 1 pour le blanc.

```
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1
1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0
1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1
1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1
1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 0 1 1
1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1
1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1
1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
```

# Fichiers image

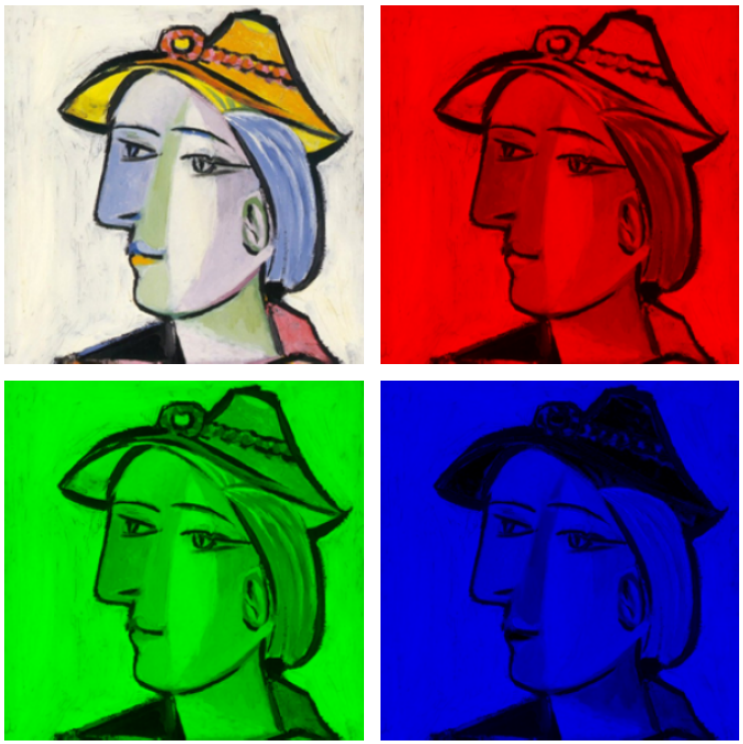
## Images en gris

Une image en gris est aussi représentée par une matrice, mais chaque élément détermine la luminance du pixel correspondant (en général un entier non signé codé sur 8 bits). Voici par exemple huit niveaux de gris différents :

# Fichiers image

## Images en couleurs

Une image en couleur peut être représentée par trois matrices, chacune déterminant la quantité respective de rouge, de vert et de bleu qui constitue l"image (c"est le modèle RGB). Les éléments de ces matrices sont des nombres entiers compris entre 0 et 255 (des entiers non signés sur 8 bits) qui déterminent la luminance de la couleur de la matrice pour le pixel correspondant.



# Fichiers image

## Lecture

Les fichiers images au format `.png`, `.jpg`, `.jpeg` sont lus, par exemple, avec la fonction imread du module misc de scipy

```
import matplotlib.pyplot as plt
from scipy import misc
image_gray = misc.imread("imgray.png")
plt.imshow(image_gray,cmap="gray")
```

pour une image en couleurs

```
import matplotlib.pyplot as plt
from scipy import misc
image_couleurs = misc.imread("imcouleurs.png")
plt.imshow(image_couleurs)
```