2019\2020

Faculté des Sciences – Kenitra

Département d'Informatique

Tableaux en Python

(Données multimédia)

Master: Master Spécialisé Génie Logiciel pour le Cloud
Présenté le: 04/Decembre/2019 (S1)
Contact: khalid.ounachad@uit.ac.ma

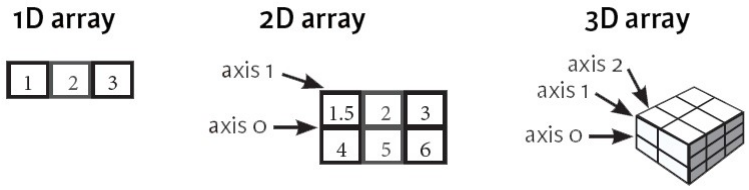
Plan

- Généralités,
- Manipulations de base(numpy , mathplotlib.pyplot),
- Images et python ,
- Son et video en python.
- Annexe

- **Généralités,**

Généralités

- **Introduction:**
- Numpy est un package pour Python spécialisé dans la manipulation des tableaux (array), pour nous essentiellement les vecteurs et les matrices.



- Les tableaux « numpy » ne gère que les objets de même type.

Généralités

- **Introduction:(par rapport aux listes)**
- Le package propose un grand nombre de routines pour un accès rapide aux données (ex. recherche, extraction), pour les manipulations diverses(ex. tri), pour les calculs (ex. calcul statistique)

• Les tableaux « numpy » sont plus performants (rapidité, gestion de la volumétrie) que les collections usuelles de Python

• Les tableaux « numpy » sont sous-jacents à de nombreux packages dédiés au calcul scientifique sous Python.

Généralités

- **Introduction:(par rapport aux listes)**
- Une matrice est un tableau (array) à 2 dimensions
- Il n'est pas possible de tout aborder dans ce support. Pour aller plus loin, voir:

https://docs.scipy.org/doc/numpy/reference/

Généralités

- **Installation:(autres modules python)**
- Pip est le gestionnaire de paquets standard pour Python . Il vous permet d'installer et de gérer des packages supplémentaires qui ne font pas partie de la bibliothèque standard Python .
- Pour se faire:

-Recherche de packages publiés dans Python Package Index (PyPI)

Généralités

- **Installation:(autres modules python)**

  -Installation de paquets supplémentaires/

  -Désinstallation des packages et de leurs dépendances.

  Voir(https://realpython.com/what-is-pip/)

  Exemple: dans C:\Windows\System32>

  pip install numpy

  pip install matplotlib

- **Manipulation de base,**

Manipulations de base

Fonctions particulières:

Commande	Description (rapide)
`np.zeros(n) ((n,p))`	vecteur nul de taille `n`, matrice nulle de taille `n,p`
`np.eye(n) ((n,p))`	matrice de taille `n` (`n,p`) avec des 1 sur la diagonale et des zéros ailleurs
`np.ones(n) ((n,p))`	vecteur de taille `n`, matrice de taille `n,p` remplie de 1
`np.diag(v)`	matrice diagonale dont la diagonale est le vecteur `v`
`np.diag(v,k)`	matrice dont la 'diagonale' décalée de `k` est le vecteur `v` (`k` est un entier relatif)
`np.random.rand(n) ((n,p))`	vecteur (taille `n`), matrice (taille `n,p`) à coefficients aléatoires uniformes sur [0,1]

Manipulations de base

Exemples:

```
>>> v=np.ones(5)
>>> v
array([ 1.,  1.,  1.,  1.,  1.])
>>> np.diag(v,2)
array([[ 0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> np.diag(v,-1)
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.]])
```

Manipulations de base

Exemples:

```
>>> b=np.ones((4,4),float)+np.eye(4,4)
>>> print b
[[ 2.  1.  1.  1.]
 [ 1.  2.  1.  1.]
 [ 1.  1.  2.  1.]
 [ 1.  1.  1.  2.]]
>>> np.reshape(b,16)
array([ 2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,
        1.,  1.,  2.])
>>> np.reshape(b,(2,8))
array([[ 2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.],
       [ 1.,  1.,  2.,  1.,  1.,  1.,  1.,  2.]])
>>> b.flatten()
array([ 2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,
        1.,  1.,  2.])
```

Manipulations de base

Création: Une première méthode consiste à convertir une liste en un tableau via la commande array:

import numpy as np

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1., 4., 5., 8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Un tableau peut être multidimensionnel ;

```
>>> a=np.array([[[1,2],[1,2]],[[1,2],[1,2]]])
>>> a
array([[[1, 2],
        [1, 2]],

       [[1, 2],
        [1, 2]]])
>>> type(a)
<type 'numpy.ndarray'>
>>> a[1,1,1]
2
```

Manipulations de base

Création:

Linspace:

```
>>> np.linspace(1., 4., 6)
array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])
```

Conversion en matrice:

```
>>> a=np.array([1, 2, 4])
>>> np.mat(a)
matrix([[1, 2, 4]])
```

Manipulations de base

Création via une saisie manuelle:

```
a = np.array([[1.2,2.5],[3.2,1.8],[1.1,4.3]])
```

Noter le rôle des [ ] et [ ] pour délimiter les portions de la matrice

```
#type de la structure
print(type(a)) #<class 'numpy.ndarray'>
#type des données
print(a.dtype) #float64
#nombre de dimensions
print(a.ndim) #2 (car c'est une matrice)
#nombre de lignes et col, shape renvoie un tuple
print(a.shape) #(3,2) → 3 lignes et 2 colonnes
#nombre totale de valeurs
print(a.size) #6, nb.lignes x nb.colonnes
```

Manipulations de base

Typage des données:

```
#print de l'ensemble
print(a)
```

→

```
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

```
#création et typage implicite
a = np.array([[1,2],[4,7]])
print(a.dtype)  #int32

#création et typage explicite – préférable !
a = np.array([[1,2],[4,7]],dtype=float)
print(a.dtype)  #float64
```

Manipulations de base

Création d'une matrice à partir d'une séquence de valeurs:

```
#création à partir d'une séquence
#attention les dim. doivent être compatibles
a = np.arange(0,10).reshape(2,5)
print(a)
```

arange() génère une séquence de valeurs, 0 à 9.
reshape() se charge de les réorganiser en matrice 2 lignes et 5 colonnes.

→

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

```
#un vecteur peut être converti en matrice
a = np.array([2.1,3.4,6.7,8.1,3.5,7.2])
print(a.shape) # (6,)
#redim. en 3 lignes x 2 col.
b = a.reshape(3,2)
print(b.shape) # (3, 2)
print(b)
```

→

```
[[ 2.1  3.4]
 [ 6.7  8.1]
 [ 3.5  7.2]]
```

```
#matrices de valeurs identiques
#ex. pour une initialisation
a = np.zeros(shape=(2,4))
print(a)
```

→

```
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
```

```
#plus généralement
a = np.full(shape=(2,4),fill_value=0.1)
print(a)
```

→

```
[[ 0.1  0.1  0.1  0.1]
 [ 0.1  0.1  0.1  0.1]]
```

Manipulations de base

Chargement à partir d'un fichier – Conversions

Les données peuvent être stockées dans un fichier texte (loadtxt pour charger, savetxt pour sauver)

matrice.txt

```
#age    poids   taille
45  62  168
34  68  187
53  85  159
```

La première ligne doit être ignorée dans ce fichier, d'où le symbole # en début de 1ère ligne.

Remarque : si besoin, modifier le répertoire par défaut avec la fonction **chdir()** du module **os** (qu'il faut importer au préalable)

```
#charger à partir d'un fichier, typage explicite
#séparateur de colonne = tabulation « \t »
a = np.loadtxt("matrice.txt",delimiter="\t",dtype=float)
print(a)
```

```
[[ 45.  62.  168.]
 [ 34.  68.  187.]
 [ 53.  85.  159.]]
```

Conversion d'une collection (type standard Python) en type array de « numpy »

```
#liste de valeurs
lst = [1.2,3.1,4.5,6.3]
print(type(lst)) # <class 'list'>
#conversion à partir d'une liste : 2 étapes asarray() et reshape()
a = np.asarray(lst,dtype=float).reshape(2,2)
print(a)
```

```
[[ 1.2  3.1]
 [ 4.5  6.3]]
```

Manipulations de base

Redimensionnement:

```
a =
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

```
#matrice de valeurs
a = np.array([[1.2,2.5],[3.2,1.8],[1.1,4.3]])
```

Accoler le vecteur **b** en tant que nouvelle ligne (axis = 0) de la matrice a

```
#ajouter une ligne – marche pour la concaténation de matrices
b = np.array([[4.1,2.6]])
c = np.append(a,b,axis=0)
print(c)
```

```
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]
 [ 4.1  2.6]]
```

Accoler le vecteur **d** en tant que nouvelle colonne (axis = 1) de la matrice a

```
#ajouter une colonne
d = np.array([[7.8],[6.1],[5.4]])
print(np.append(a,d,axis=1))
```

```
[[ 1.2  2.5  7.8]
 [ 3.2  1.8  6.1]
 [ 1.1  4.3  5.4]]
```

Insertion de **b** en tant que nouvelle ligne (axis = 0) à la position n°1

```
#insertion
print(np.insert(a,1,b,axis=0))
```

```
[[ 1.2  2.5]
 [ 4.1  2.6]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

Suppression de la ligne (axis = 0) via son indice (n°1)

```
#suppression
print(np.delete(a,1,axis=0))
```

```
[[ 1.2  2.5]
 [ 1.1  4.3]]
```

Redimensionnement d'une matrice

```
#modifier la dimension d'une matrice existante
#parcourt les données lignes par ligne
h = np.resize(a,new_shape=(2,3))
print(h)
```

```
[[ 1.2  2.5  3.2]
 [ 1.8  1.1  4.3]]
```

Manipulations de base

Statistiques:

Principe : les calculs sont réalisés selon un axe (0 : traitement des valeurs en ligne pour chaque colonne, 1 : inversement)

v =
```
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

```
#moyenne des lignes pour chaque colonne
print(np.mean(v,axis=0)) # [1.833  2.867]

#moyenne des colonnes pour chaque ligne
print(np.mean(v,axis=1)) # [1.85  2.5  2.7]

#somme cumulée des valeurs en ligne pour chaque colonne
print(np.cumsum(v,axis=0))
```

```
[[ 1.2  2.5]
 [ 4.4  4.3]
 [ 5.5  8.6]]
```

```
#matrice de corrélation
#rowvar = 0 pour indiquer que les variables
#sont organisés en colonnes
m = np.corrcoef(v,rowvar=0)
print(m)
```

```
[[ 1.          -0.74507396]
 [-0.74507396  1.        ]]
```

Manipulations de base

Calcul matriciel et algèbre linéaire:

x =
```
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

y =
```
[[ 2.1  0.8]
 [ 1.3  2.5]]
```

```
#transposition
print(np.transpose(x))
```

```
[[ 1.2  3.2  1.1]
 [ 2.5  1.8  4.3]]
```

```
#multiplication
print(np.dot(x,y))
```

```
[[  5.77   7.21]
 [  9.06   7.06]
 [  7.9   11.63]]
```

```
#déterminant
print(np.linalg.det(y))
```

4.21

```
#inversion
print(np.linalg.inv(y))
```

```
[[ 0.59382423 -0.19002375]
 [-0.3087886   0.49881235]]
```

Manipulations de base

Calcul matriciel et algèbre linéaire:

```
x =  [[ 1.2  2.5]
      [ 3.2  1.8]
      [ 1.1  4.3]]

y =  [[ 2.1  0.8]
      [ 1.3  2.5]]
```

Solution de Y.a = z

```
#résolution d'équation
z = np.array([1.7,1.0])
print(np.linalg.solve(y,z)) # [0.8195  -0.0261]
```

On peut faire $a = Y^{-1}.z$

```
#vérification
print(np.dot(np.linalg.inv(y),z)) # [0.8195  -0.0261]
```

```
#matrice symétrique avec XᵀX
s = np.dot(np.transpose(x),x)
print(s)
```

```
[[ 12.89  13.49]
 [ 13.49  27.98]]
```

```
#val. et vec. propres d'une matrice symétrique
print(np.linalg.eigh(s))
```

```
(array([  4.97837925,  35.89162075]),
array([[-0.86259502,  0.50589508],
       [ 0.50589508,  0.86259502]]))
```

Manipulations de base

Tracé de courbes graphiques:

Commande	Bibl.	Résultat
`plot(x,y,c?)`	matplotlib.pyplot	Affiche les points définis par les vecteurs x et y, (Option~: c permet de définir le format et la couleur du tracé)
`imshow(m,c?)`	matplotlib.pyplot	Affiche la matrice m en deux dimensions
`show()`	matplotlib.pyplot	Affiche la figure courante
`savefig(name)`	matplotlib.pyplot	Sauvegarde la figure courante dans le fichier name
`clf()`	matplotlib	Efface la figure courante
`legend(array,loc?)`	matplotlib.pyplot	Dessine une légende contenant les lignes apparaissant dans array (Option~: $loc = (x, y)$ pour définir l'emplacement)
`xlabel(str) ylabel(str)`	matplotlib.pyplot	Imprime une légende pour décrire les axes horizontaux et verticaux
`axis([xl,xr,yb,yt])`	matplotlib.pyplot	Cadre la figure sur le rectangle décrit par les 4 coordonnées.

Manipulations de base

Tracé de courbes graphiques: Exemple

Commande	Bibl.	Résultat
plot(x,y,c?)	matplotlib.pyplot	Affiche les points définis par les vecteurs x et y, (Option~: c permet de définir le format et la couleur du tracé)
imshow(m,c?)	matplotlib.pyplot	Affiche la matrice m en deux dimensions
show()	matplotlib.pyplot	Affiche la figure courante
savefig(name)	matplotlib.pyplot	Sauvegarde la figure courante dans le fichier name
clf()	matplotlib	Efface la figure courante
legend(array,loc?)	matplotlib.pyplot	Dessine une légende contenant les lignes apparaissant dans array (Option~: $loc = (x, y)$ pour définir l'emplacement)
xlabel(str) ylabel(str)	matplotlib.pyplot	Imprime une légende pour décrire les axes horizontaux et verticaux
axis([xl,xr,yb,yt])	matplotlib.pyplot	Cadre la figure sur le rectangle décrit par les 4 coordonnées.

Manipulations de base

Tracé de courbes graphiques: Exemple

Le module Matplotlib est chargé de tracer les courbes :

```
>>> import matplotlib.pyplot as plt
```

Commençons par la fonction sinus.

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-5,5,100)
plt.plot(x,np.sin(x)) # on utilise la fonction sinus de Numpy
plt.ylabel('fonction sinus')
plt.xlabel("l'axe des abcisses")
plt.show()
```

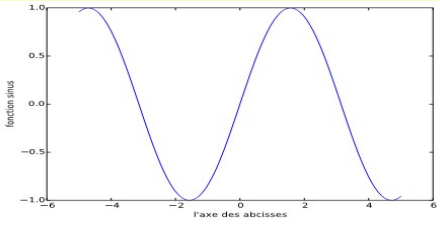


Figure1 : Graphe de la fonction sinus

Manipulations de base

Tracé de courbes graphiques: Exemple2

```
>>> x=np.linspace(-np.pi,np.pi,100)
>>> plt.plot(x,np.sin(x),color="red", linewidth=2.5, linestyle="-", label="sinus")
>>> plt.plot(x,np.cos(x),color="blue", linewidth=2.5, linestyle="-", label="cosinus")
>>> plt.legend(loc='upper left')
>>> plt.axis([-np.pi,np.pi,-1,1])
>>> plt.yticks([-1, 0, +1],
[r'$-1$', r'$0$', r'$+1$'])
>>> plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
[r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
```

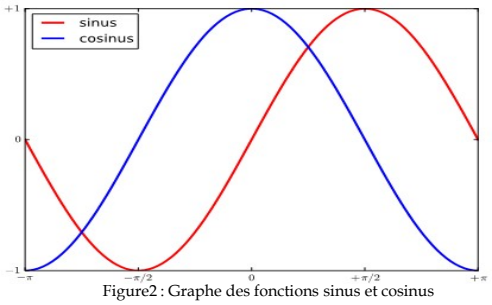


Figure2 : Graphe des fonctions sinus et cosinus

- **Images et python,**

Images et Python

Principe:

l'objectif sera toujours de transformer une image en tableau numpy, pour pouvoir ensuite la manipuler.

Les modules:

- **matplotlib**
- **pil**
- **Open cv**
- **…**

Images et Python

Les modules:

PIL permet de lire une image enregistrée localement dans de nombreux formats.

matplotlib.image ne permet que de charger des .png. Mais vous avez directement un tableau numpy (pas besoin de transformation).

Images et Python

Lecture de l'image:

Avec "matplotlib.image"

```
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread("monimage.png")
```

Avec PIL

```
from PIL import Image import numpy as np
imgpil = Image.open("monimage.png") #
anciennement np.asarray img =
np.array(imgpil) # Transformation de
l'image en tableau numpy
```

Images et Python

Affichage de l'image :

Matplotlib permet d'afficher une image (si c'est un tableau numpy).

Avec "matplotlib.image"

```
import matplotlib.pyplot as plt
plt.imshow(img)
plt.show()
```

Avec PIL

```
from PIL import Image
im = Image.open("monimage.jpg")
im.show()
```

Images et Python

Sauvgarde de l'image: Pour enregistrer des images, on utilise une méthode similaire au chargement. Matplotlib permet ainsi de sauvegarder directement un tableau numpy au format PNG uniquement. PIL permettra de sauvegarder dans n'importe quel format, pourvu qu'on ait transformé le tableau numpy en Image PIL.

Avec Matplotlib

```
import matplotlib.image as mpimg
mpimg.imsave("resultat.png", img)
```

Avec PIL

Avec PIL, il faut s'abord transformer le tableau numpy en image PIL.

```
from PIL import Image
imgpil = Image.fromarray(img) # Transformation du tableau en image PIL
imgpil.save("resultat.jpg")
```

- **Son et Vidéo en Python,**

Son et Python

Introduction:

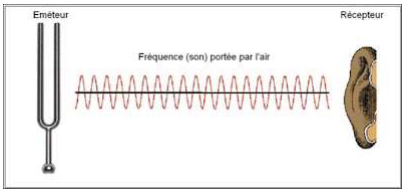
Le son est une sensation auditive provoquée par une vibration.

Trois éléments sont nécessaires à l'existence d'un son :

-une source qui produit le son,

-un milieu qui transmet la vibration,

-un récepteur : l'oreille.

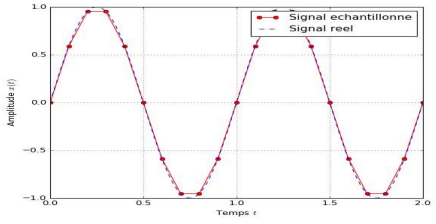


Son et Python

Principe:

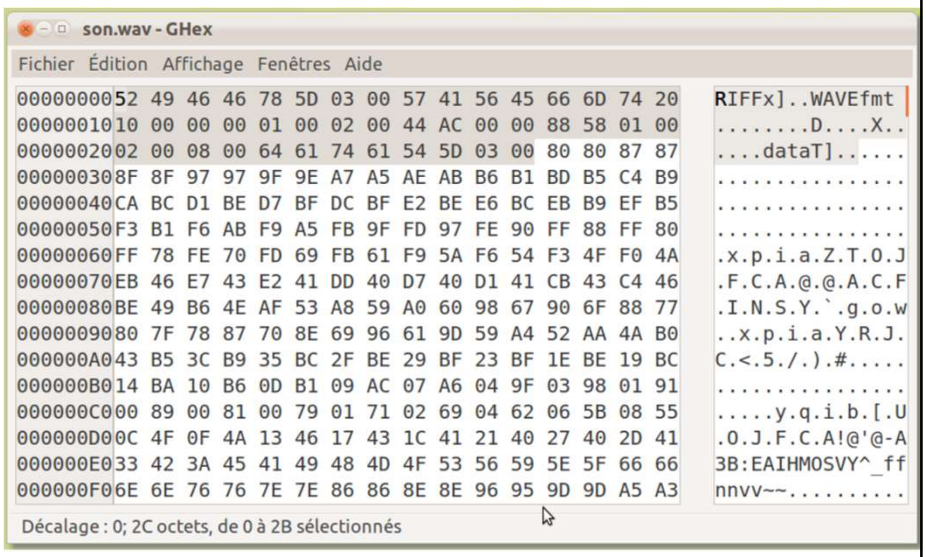
Un signal unidimensionnel peut être vu comme une fonction mathématique du type: $x:t \mapsto x(t)$

```
import numpy as np import matplotlib.pyplot as plt
# Signal
T = 1.
def signal(t): return np.sin(2. * np.pi * t / T)
# Echantillonnage
D = 2. # Duree d'observation
fe = 10. # Frequence d'echantillonnage
N = (D * fe) + 1 # Nombre de points enregistres
te = np.linspace(0., (N-1)/fe, N) # Grille d'echantillonnage
tp = np.linspace(0., D, 1000) # Grille plus fine pour tracer l'allure du signal parfait # Trace du signal
plt.plot(te, signal(te), 'or-', label = u"Signal echantillonne")
plt.plot(tp, signal(tp), 'b--', label = u"Signal reel")
plt.grid() plt.xlabel("Temps $t$")
plt.ylabel("Amplitude $x(t)$")
plt.legend()
plt.show()
```



Son et Python

Contenu: édition hexadécimal



Son et Python

Modules:

- Pyaudio,
- Winsound,
- Pygame,
- Pyglet,
- Pymedia,
- espeak
- cv
- etc.

Son et Python

Exemples: Ecouter le son

```
import pygame
pygame.mixer.init()
pygame.mixer.Sound("son.wav").play()
while pygame.mixer.get_busy():
    # lecture en cours
    pass
```

```
# lecture audio (sortie vers la carte son)
import winsound
winsound.PlaySound('son.wav',winsound.SND_FILENAME)
```

Vidéo et Python

Définition:

La vidéo est une succession d'images animées défilant à une certaine cadence afin de créer une illusion de mouvement pour l'œil humain.

Elle peut être:

- analogique (signal continu d'intensité de luminance)
- ou numérique (suite de trames ou images)

En vidéo, c'est la caméra qui transforme l'information lumineuse (photons) en signal électrique (électrons). En vidéo analogique, l'intensité de ce signal électrique varie de façon continue.

Vidéo et Python

Exemples: Le programme pour lire une webcam branchée sur un port USB

```
import numpy as np
import cv2 as cv
cap = cv.VideoCapture(0)
while(True):
    # Capture image par imaghe
    ret, img = cap.read()
    # Préparation de l'affichage de l'image
    cv.imshow('frame',img)
    # affichage et saisie d'un code clavier
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
# Ne pas oublier de fermer le flux et la fenetre
cap.release()
cv.destroyAllWindows()
```

Vidéo et Python

Changement de la taille

Pour changer la taille on utilise la méthode set, après avoir ouvert le flux video

```
cap.set(cv.CAP_PROP_FRAME_WIDTH,1280)
cap.set(cv.CAP_PROP_FRAME_HEIGHT,960)
```

- Lecture d'une webcam et enregistrement d'une vidéo

```
import numpy as np
import cv2 as cv
webcam = cv.VideoCapture(0)
fourcc = cv.VideoWriter_fourcc(*'MJPG')
fichier = cv.VideoWriter('c:/temp/test.avi',fourcc,20,(640,480))
while(True):
    # Capture image par image
    ret, img = webcam.read()
    if ret==True:
        fichier.write(img)
        # Preparation de l'affichage de l'image
        cv.imshow('Ma Webcam',img)
        # affichage et saisie d'un code clavier
        if cv.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
# Ne pas oublier de fermer le flux et la fenetre
webcam.release()
fichier.release()
cv.destroyAllWindows()
```

Vidéo et Python

Changement de la taille

Pour changer la taille on utilise la méthode set, après avoir ouvert le flux vidéo

```
cap.set(cv.CAP_PROP_FRAME_WIDTH,1280)
cap.set(cv.CAP_PROP_FRAME_HEIGHT,960)
```

- Lecture d'une webcam et enregistrement d'une vidéo

```
import numpy as np
import cv2 as cv
webcam = cv.VideoCapture(0)
fourcc = cv.VideoWriter_fourcc(*'MJPG')
fichier = cv.VideoWriter('c:/temp/test.avi',fourcc,20,(640,480))
while(True):
    # Capture image par image
    ret, img = webcam.read()
    if ret==True:
        fichier.write(img)
        # Preparation de l'affichage de l'image
        cv.imshow('Ma Webcam',img)
        # affichage et saisie d'un code clavier
        if cv.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
# Ne pas oublier de fermer le flux et la fenetre
webcam.release()
fichier.release()
cv.destroyAllWindows()
```

Vidéo et Python

- Lecture d'une webcam et enregistrement d'une vidéo


```
import numpy as np
import cv2 as cv
webcam = cv.VideoCapture(0)
fourcc = cv.VideoWriter_fourcc(*'MJPG')
fichier = cv.VideoWriter('c:/temp/test.avi',fourcc,20,(640,480))
while(True):
    # Capture image par image
    ret, img = webcam.read()
    if ret==True:
        fichier.write(img)
        # Preparation de l'affichage de l'image
        cv.imshow('Ma Webcam',img)
        # affichage et saisie d'un code clavier
        if cv.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
# Ne pas oublier de fermer le flux et la fenetre
webcam.release()
fichier.release()
cv.destroyAllWindows()
```

Vidéo et Python

- Lecture d'un fichier vidéo: La lecture d'un fichier vidéo ou du flux d'une webcam fonctionne sur le même principe. La différence est dans l'origine : pour un fichier il faut donner le nom !

```
import numpy as np
import cv2 as cv
fichier = cv.VideoCapture('c:/temp/test.avi')
while(True):
    # Capture image par image
    ret, img = fichier.read()
    if ret==True:
        # Preparation de l'affichage de l'image
        cv.imshow('Mon fichier',img)
        # affichage et saisie d'un code clavier
        if cv.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
# Ne pas oublier de fermer le flux et la fenetre
fichier.release()
cv.destroyAllWindows()
```

Annexe

Dialoguer avec Excel en Python : openpyxl:

```
from openpyxl import Workbook
from openpyxl import load_workbook

#ouvrir le fichier excel sample.xlsx (le fichier excel doit
#se trouver dans le même répertoire que le fichier python)
wb = load_workbook(filename = "sample.xlsx")

#charger la feuille excel active dans une variable python
ws = wb.active

#on récupère la valeur de la case B2 puis on l'affiche
a = ws.cell(row=2, column=2).value
print(a)

#on écrit la valeur de la variable b dans la case B1
b = 425.5
ws.cell(row=1, column=2).value = b

#on enregistre le fichier sample.xlsm
wb.save("sample.xlsx")

#ouvrir un nouveau fichier excel
wb2 = Workbook()

#charger la feuille excel active dans une variable python
ws2 = wb2.active

#on écrit la valeur de la variable b dans la case A2
ws2.cell(row=1, column=2).value = b

#on enregistre le nouveau fichier excel en lui donnant le nom sample2.xlsx
wb2.save("sample2.xlsx")
```

MERCI


جامعة ابن طفيل
UNIVERSITÉ IBN TOFAÏL
Faculté des Sciences
MASTER D'UNIVERSITÉ SPÉCIALISÉ
.:: Informatique ::.
GÉNIE LOGICIEL POUR LE CLOUD
Exercez les métiers de demain
BIG DATA
Business Intelligence
Big Data
Cloud Computing
Coordinateur
Pr. Moulay Youssef Hadi
Département d'informatique
hadiyoussef@gmail.com
0694 500 534

Travaux Pratiques:

- TP5