Traitement d'images matricielles en Python

Sommaire

L'objectif est de présenter une application peu connue des matrices aux enseignants de lycées : le traitement des images numériques. (Extrait de l'article Matrices et Images Numériques d'Antoine Nectoux)

1. Python

1.1. Présentation

Python est un langage de programmation interprété.

Les programmes Python sont des scripts donc des fichiers textes qui contiennent le code source. Ils ont besoin de l'interpréteur Python pour s'exécuter.

Pour programmer en Python, on a besoin des ressources suivantes :

- un interpréteur Python installé sur sa machine
- un éditeur de texte ou un EDI pour produire des programmes Python

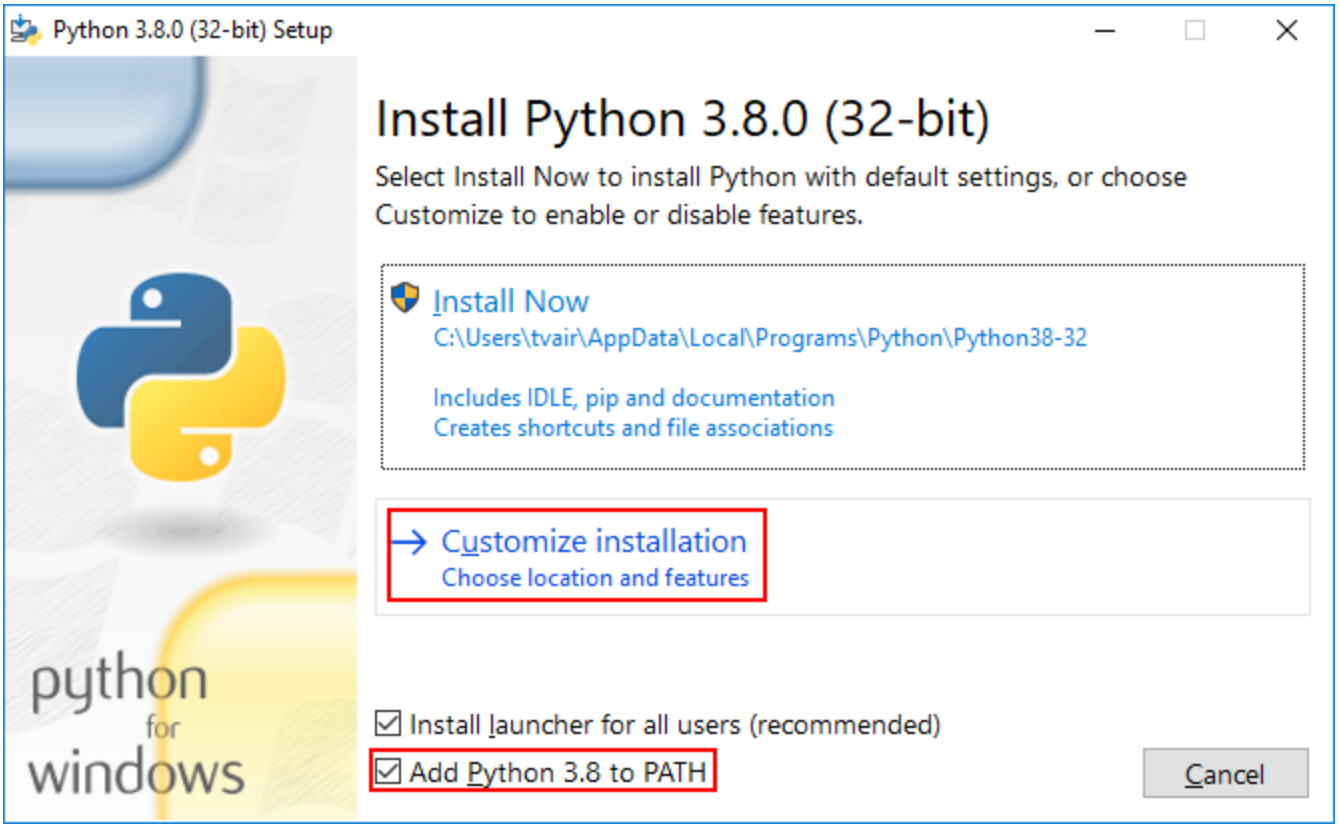
Liens :

- Site officiel
- Documentation Python3
- Les bases du langage Python
- Exercices

1.2. Installation sous Windows

Lien : Télécharger Python3

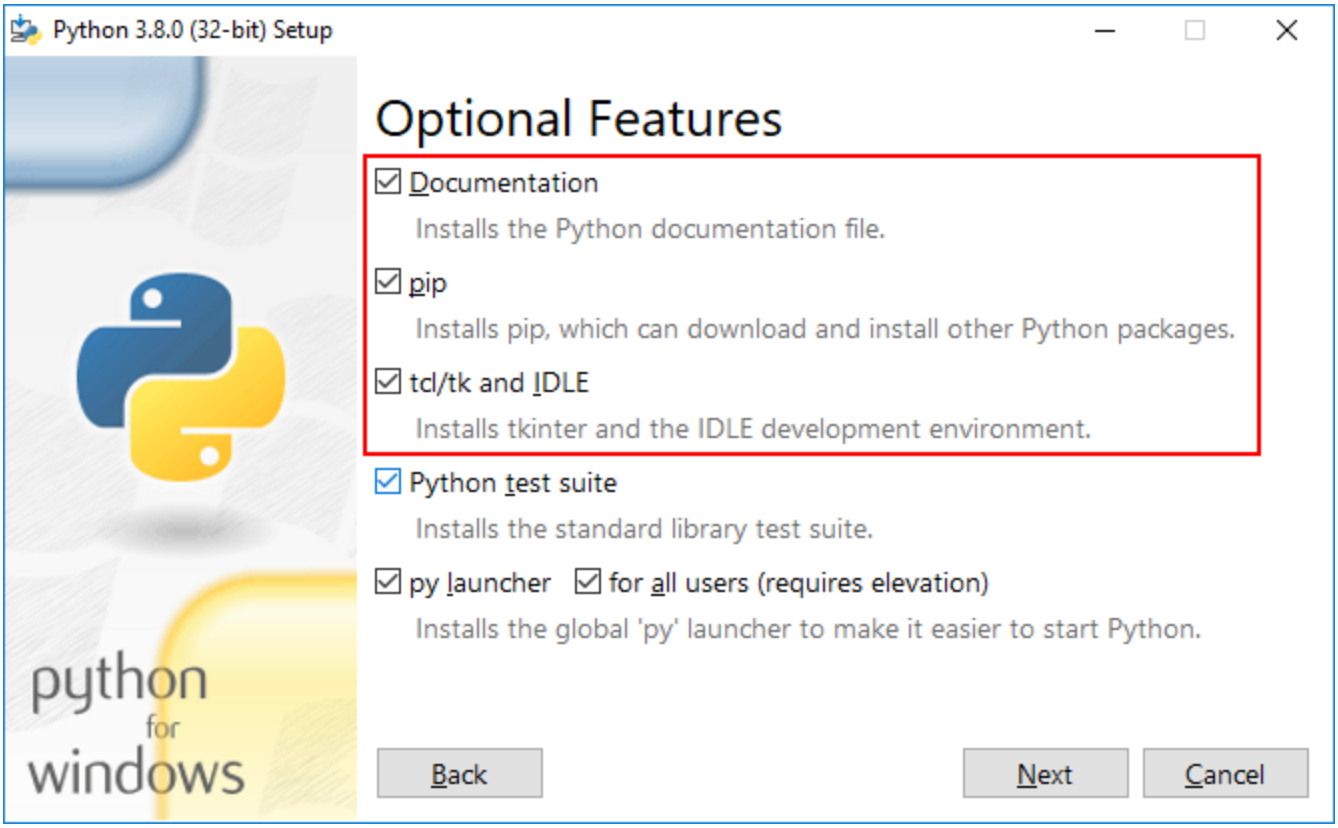
Exécuter l'installateur :



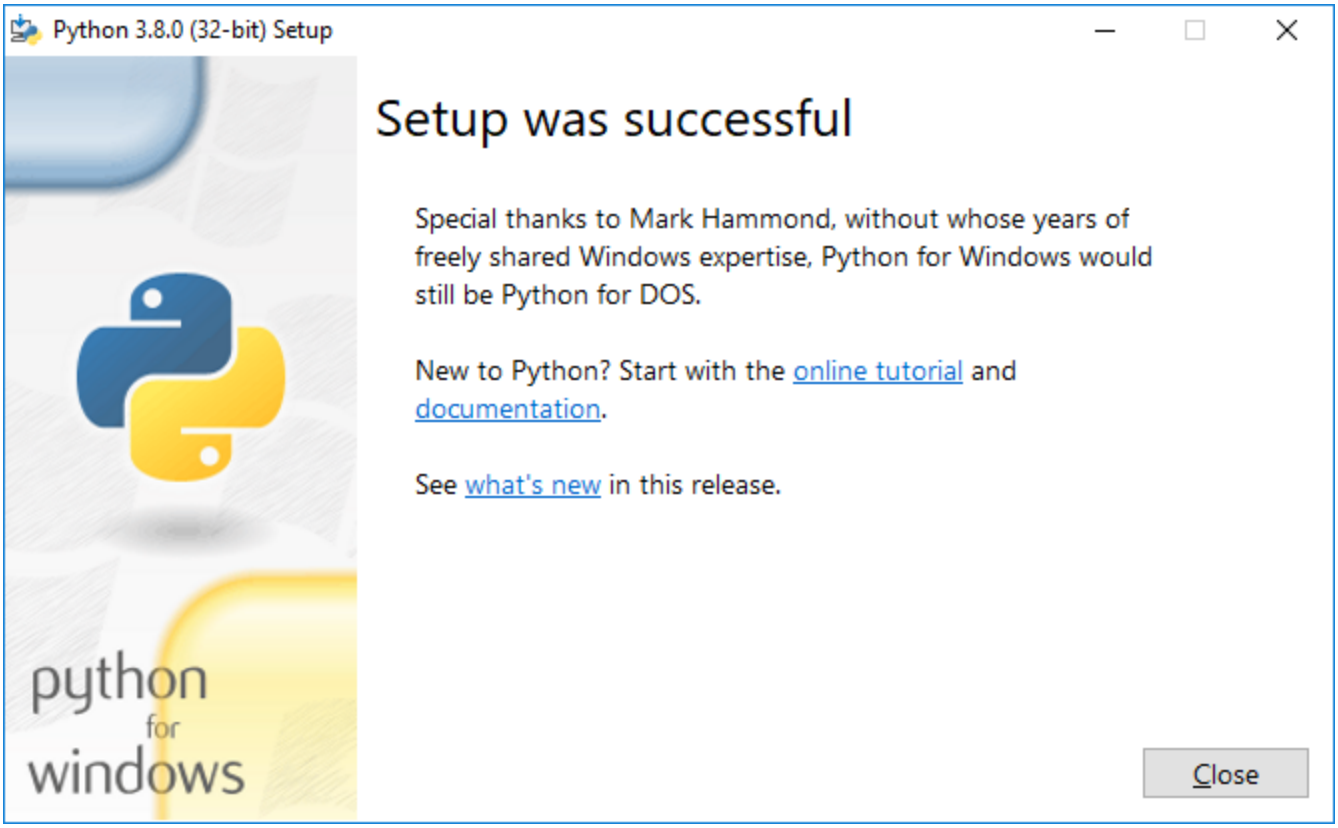
`pip` est un utilitaire qui permettra d'installer des modules Python

`tkinter` est une bibliothèque permettant de créer des interfaces graphiques utilisateur (GUI en Anglais)

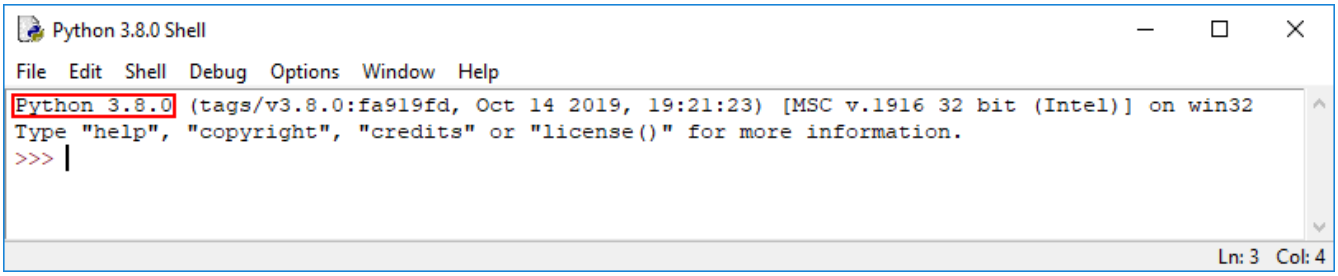
Il est intéressant d'ajouter le chemin complet vers l'interpréteur Python dans la variable `PATH`. La commandes `python` sera alors accessible à partir de l'interpréteur de commandes (`cmd.exe`)



On démarre l'installation …



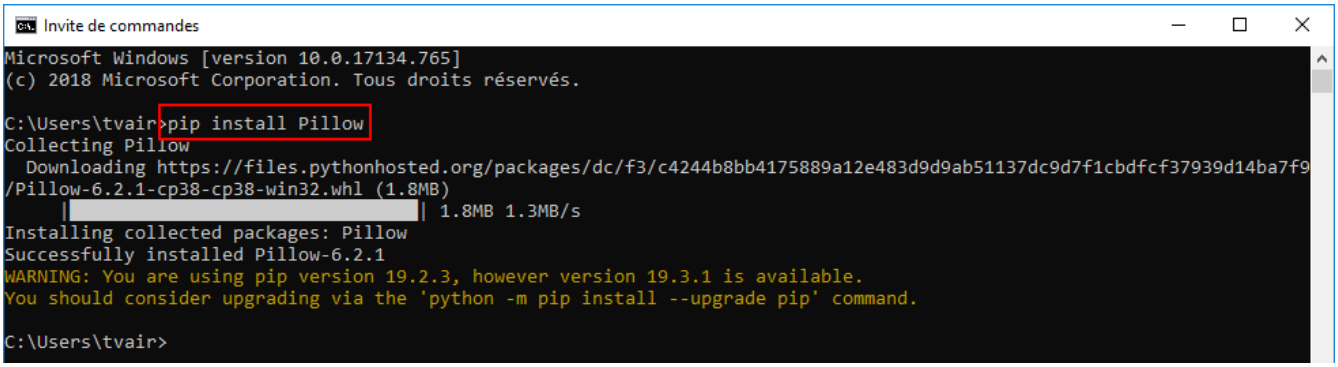
Vérifions :



Il est possible de programmer directement dans le *shell* Python.

1.3. Bibliothèque PIL (*Python Imaging Library*)

On installe maintenant le *package* Pillow utilisé dans les exemples de traitement d'images ci-dessous. Pour cela, le plus simple est de démarrer l'interpréteur de commandes `cmd.exe` puis de saisir la commande `pip install Pillow` :



PIL (*Python Imaging Library*) est une bibliothèque de traitement d'image (cf. Documentation).

2. Activités de traitement d'images

Les activités présentés ici ont pour objectif de montrer l'utilisation des matrices dans le traitement d'images numériques.

En effet, une image numérique (au niveau d'un périphérique comme l'écran) est une **matrice de pixels** (*px*). On parle d'image matricielle.

Pour une **image binaire** (ou image en noir et blanc), les éléments de l'image sont représentés par des pixels noirs (des `0` par exemple) et des pixels blancs (donc des `1`).

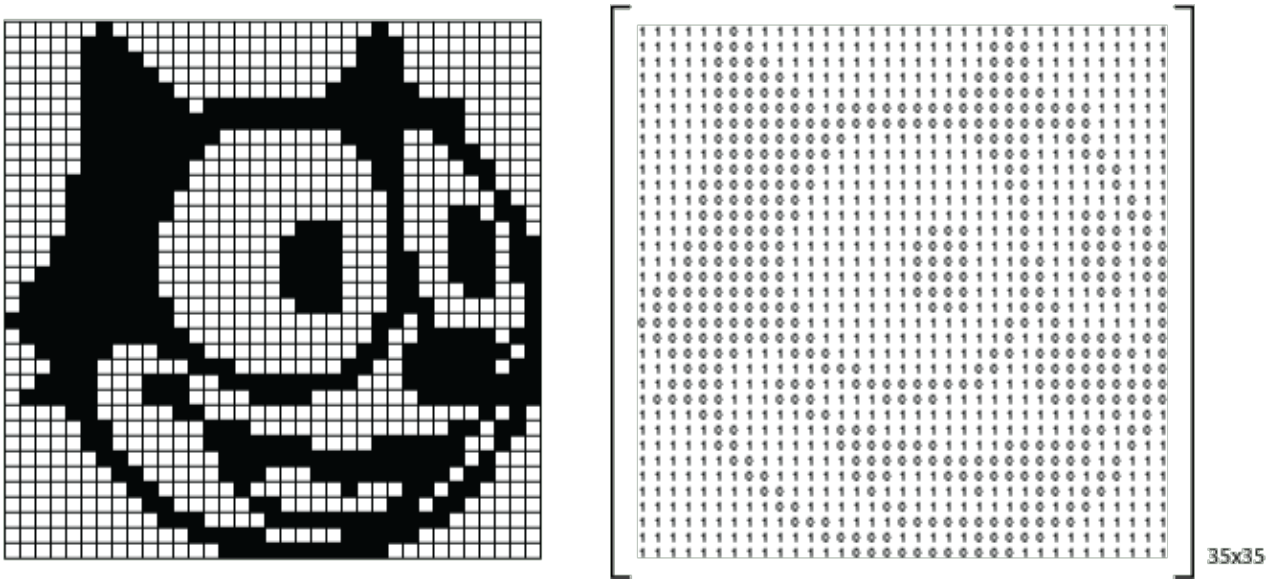
Pour une image en couleurs, chaque pixel est associée une couleur, usuellement décomposée en trois composantes primaires par synthèse additive : rouge vert bleu (RVB ou **RGB** en anglais).

Un pixel est donc codé sur un ou plusieurs bits, par exemple :

- 1 bit : `2^1 = 2` couleurs (noir et blanc par exemple)
- 8 bits : `2^8 = 256` couleurs (256 niveaux de gris de `0` pour le noir à `255` pour le blanc)
- 24 bits : `2^24 = 16 777 216` couleurs (3 x 8 bits pour les composantes Rouge Vert Bleu)

Par exemple, la petite image de **Félix le Chat** peut être représentée par une matrice de taille `35 \times 35` :

La matrice correspondant à Felix le Cat



Lire l'article d'Antoine Nectoux : http://blog.kleinproject.org/?p=719&lang=fr

Les fichiers peuvent enregistrer l'image sous forme d'un tableau des valeurs représentant les pixels (format BMP par exemple), éventuellement comprimé (formats PNG ou JPEG), ou bien sous forme vectorielle (format SVG par exemple), c'est-à-dire sous forme d'instructions permettant de reconstituer une image matricielle. Dans ce cas, la couleur peut être également enregistrée sous forme vectorielle, avec comme paramètres (teinte, saturation, lumière).

2.1. Activité n°1 : manipuler une image

Ce premier programme effectue les opérations basiques suivantes :

- afficher le format, la taille et le mode d'une image
- afficher les dimensions d'une image
- afficher une image
- calculer et afficher le nombre de pixels d'une image
- modifier la taille d'une image et en obtient une nouvelle
- afficher la résolution et la taille de l'écran
- afficher la résolution et la taille de l'image en ppp (en pixels par pouce) ou ppi (*pixel per inch*)

Le programme Python ci-dessous est commenté afin d'être certain d'utiliser le bon vocabulaire lié à la programmation Python.

```
# coding: utf-8

# Python3 pour Windows : https://www.python.org/downloads/
# PIL (Python Imaging Library) (Pillow fork)
# Installation : pip install Pillow
# Documentation : https://pillow.readthedocs.io/en/stable/

print("ACTIVITÉ 1")

# PIL est une bibliothèque de traitement d'image
import PIL;
print("PIL version ", PIL.__version__)

# on importe le module Image de la bibliothèque PIL
# le module Image fournit une classe du même nom qui sera utilisée pour représenter
une image.
from PIL import Image

# cf. https://fr.wikipedia.org/wiki/Lenna

# Image est une classe (c'est un type Image)
# img est un objet de la classe Image
# la création d'un objet se nomme l'instanciation
# open() est une fonction (méthode) de la classe Image qui permet de charger un
```

```
fichier image
img = Image.open("./images/lena.jpg");
# format, size et mode sont des données (attributs) de la classe Image
print(img.format, img.size, img.mode)
# exemple : les modes courants sont "L" (luminance) pour les images en niveaux de
gris, "RGB" pour les images en couleurs vraies et "CMYK" pour les images pour
l'imprimerie (quadrichromie)
print("Fichier :", img.filename)

print("2. Calculez le nombre de pixels de cette image.")

# Affichage des dimensions de l'image
print("L =", img.width, "x H =", img.height)
# Calcul et Affichage du nombre de pixels de l'image
print("Nombre de pixels =", (img.width*img.height) , "pixels")

# Affichage de l'image
img.show()

# resize() est une fonction (méthode) de la classe Image qui permet de modifier la
taille d'une image et d'en obtenir une nouvelle
# im2 est un objet de la classe Image
im2 = img.resize((228,192))
# save() est une fonction (méthode) de la classe Image qui sauvegarder l'image dans un
fichier (ici, l'extension précisera le format de l'image)
im2.save("./images/lena_228x192.png")
print(im2.format, im2.size, im2.mode)
print("Nombre de pixels =", "L =", im2.width, "x H =", im2.height, "=",
(im2.width*im2.height) , "pixels")
im2.show()

# size est un tuple (les tuples sont des séquences qu'on ne pourra plus modifier)
size = (144,96) = les parenthèses ne sont pas obligatoires
width, height = size
print(size, width, height)

im3 = img.resize(size)
im3.save("./images/lena_144x96.png")
print(im3.format, im3.size, im3.mode)
print("Nombre de pixels =", "L =", im3.width, "x H =", im3.height, "=",
(im3.width*im3.height) , "pixels")
im3.show()

#import Tkinter as tk = en python 2
import tkinter as tk = en python 3
ecran = tk.Tk()
#print(ecran.winfo_screen()) = nom de l'écran

print("1. Calculez la résolution de l'écran en pixels par centimètre, puis en ppp.")

# Résolution = Nombre de pixels en largeur / Largeur
```

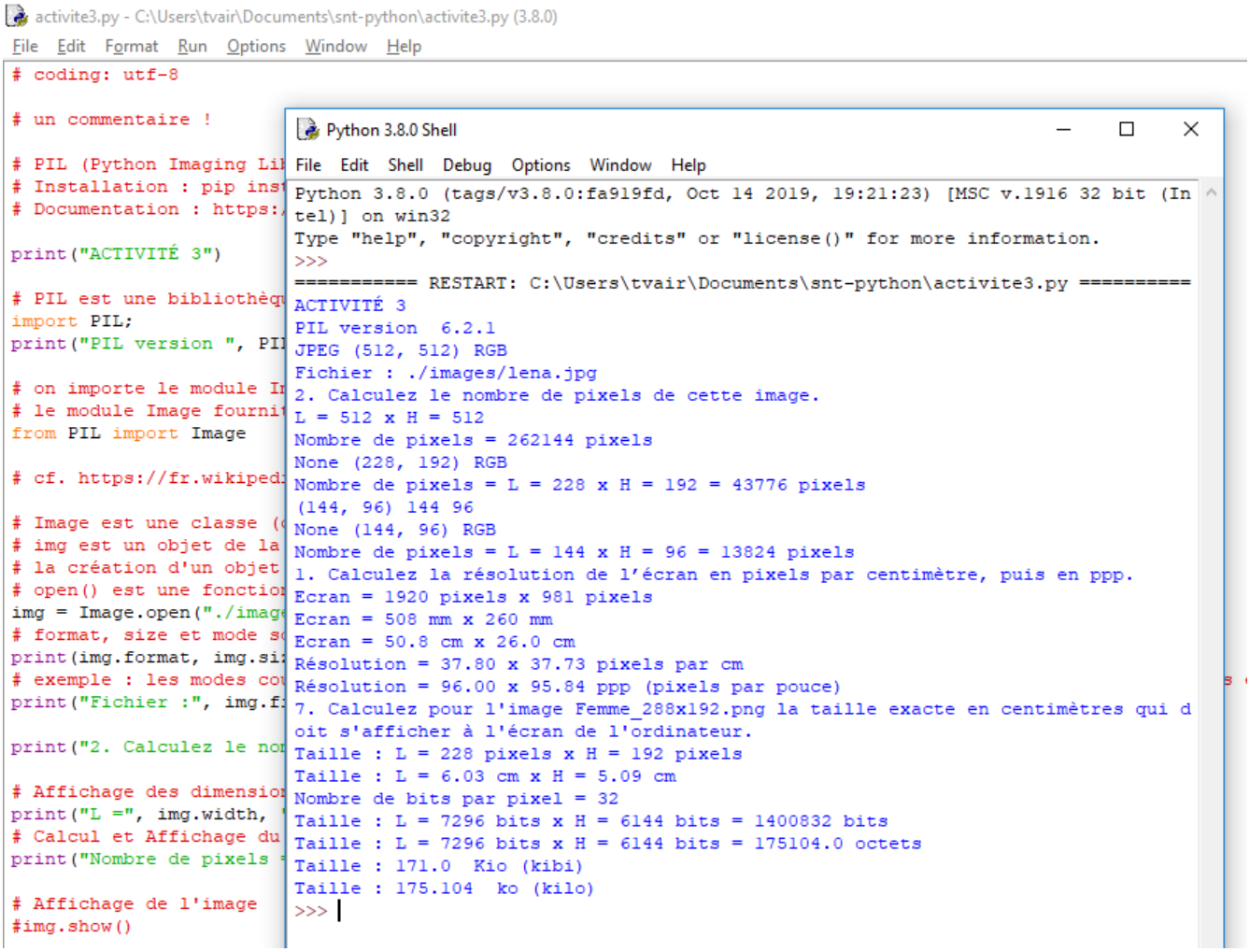
```
w, h = ecran.winfo_screenwidth(), ecran.winfo_screenheight()
print("Ecran = %s pixels x %s pixels" % (w, h))

# Taille de l'écran
print("Ecran = %s mm x %s mm" % (ecran.winfo_screenmmwidth(),
ecran.winfo_screenmmheight()))
print("Ecran = %s cm x %s cm" % (ecran.winfo_screenmmwidth()/10,
ecran.winfo_screenmmheight()/10))

# Résolution (en pixels par cm)
ppc_w = (w / (ecran.winfo_screenmmwidth()/10))
ppc_h = (h / (ecran.winfo_screenmmheight()/10))
print("Résolution = %.2f x %.2f pixels par cm" % (ppc_w, ppc_h))

# Résolution ppp (en pixels par pouce) ou ppi (pixel per inch)
ppi_w = (w / (ecran.winfo_screenmmwidth()/10)) * 2.54
ppi_h = (h / (ecran.winfo_screenmmheight()/10)) * 2.54
print("Résolution = %.2f x %.2f ppp (pixels par pouce)" % (ppi_w, ppi_h))

print("2. Calculez pour l'image Femme_288x192.png la taille exacte en centimètres qui
doit s'afficher à l'écran de l'ordinateur.")

print("Taille : L =", im2.width, "pixels x H =", im2.height, "pixels")
print("Taille : L = %.2f" % (im2.width/ppc_w), "cm x H = %.2f" % (im2.height/ppc_h),
"cm")

# Bonus :
depth = ecran.winfo_screendepth() = retourne le nombre de bits par pixel
print("Nombre de bits par pixel =", depth) = RGB 24 bits = R8 + G8 + B8

# cf. le format BMP (sans compression)
print("Taille : L =", (im2.width*depth), "bits x H =", (im2.height*depth), "bits",
"=", ((im2.width*im2.height*depth)), "bits")
print("Taille : L =", (im2.width*depth), "bits x H =", (im2.height*depth), "bits",
"=", ((im2.width*im2.height*depth))/8, "octets")
# https://fr.wikipedia.org/wiki/Préfixe_binaire
print("Taille :", (((im2.width*im2.height*depth))/8)/1024, " Kio (kibi)")
print("Taille :", (((im2.width*im2.height*depth))/8)/1000, " ko (kilo)")
#print("Taille :", (((im2.width*im2.height*depth))/8)/1024/1024, " Mio (mebi)")
#print("Taille :", (((im2.width*im2.height*depth))/8)/1000/1000, " Mo (mega)")
```

L'image utilisée ici est célèbre : Lenna

Il est possible d'exécuter ce programme directement dans l'interpréteur de commandes `cmd.exe` :

```
Invite de commandes
 Le volume dans le lecteur C n'a pas de nom.
 Le numéro de série du volume est 183A-4786

 Répertoire de C:\Users\tvair\Documents\snt-python

24/11/2019  10:11    <DIR>          .
24/11/2019  10:11    <DIR>          ..
24/11/2019  10:00             5 956 activite3.py
19/10/2019  08:10               511 helloworld2.py
19/10/2019  08:38               499 helloworld3.py
24/11/2019  10:11    <DIR>          images
               3 fichier(s)            6 966 octets
               3 Rép(s)   5 860 737 024 octets libres

C:\Users\tvair\Documents\snt-python>python activite3.py
ACTIVITÉ 3
PIL version  6.2.1
JPEG (512, 512) RGB
Fichier : ./images/lena.jpg
2. Calculez le nombre de pixels de cette image.
L = 512 x H = 512
Nombre de pixels = 262144 pixels
None (228, 192) RGB
Nombre de pixels = L = 228 x H = 192 = 43776 pixels
(144, 96) 144 96
None (144, 96) RGB
Nombre de pixels = L = 144 x H = 96 = 13824 pixels
1. Calculez la résolution de l'écran en pixels par centimètre, puis en ppp.
Ecran = 1920 pixels x 981 pixels
Ecran = 508 mm x 260 mm
Ecran = 50.8 cm x 26.0 cm
Résolution = 37.80 x 37.73 pixels par cm
Résolution = 96.00 x 95.84 ppp (pixels par pouce)
7. Calculez pour l'image Femme_288x192.png la taille exacte en centimètres qui doit s'afficher à l'écran de l'ordinateur
.
Taille : L = 228 pixels x H = 192 pixels
Taille : L = 6.03 cm x H = 5.09 cm
Nombre de bits par pixel = 32
Taille : L = 7296 bits x H = 6144 bits = 1400832 bits
Taille : L = 7296 bits x H = 6144 bits = 175104.0 octets
Taille : 171.0  Kio (kibi)
Taille : 175.104  ko (kilo)

C:\Users\tvair\Documents\snt-python>
```

Ou à partir de l'IDE fourni par l'installateur Python en cliquant sur `Run` :



Quelques commandes sous GNU/Linux qui permettent d'obtenir la taille en octets (suivant le préfixe utilisé) :

```
$ ls -l images/lena_228x192.data
-rw-rw-r-- 1 tv tv 131328 nov.  24 09:23 images/lena_228x192.bmp

$ ls -lh images/lena_228x192.data
-rw-rw-r-- 1 tv tv 129K nov.  24 09:23 images/lena_228x192.bmp

$ ls -l --si images/lena_228x192.data
-rw-rw-r-- 1 tv tv 132k nov.  24 09:23 images/lena_228x192.bmp
```

Lien : kibi vs kilo

2.2. Activité n°2 : binariser une image

En traitement d'images, la binarisation est une opération qui produit une image ayant deux classes de pixels, on parle alors d'une **image binaire**. En général, les pixels sont représentés par des pixels noirs et des pixels blancs dans une image binaire. Il existe deux types de technique de binarisation :

- Binarisation par seuillage
- Binarisation par segmentation

La différence entre la binarisation et la segmentation est que la binarisation produit toujours deux classes alors que la segmentation peut en produire plusieurs. Cependant, ces deux termes sont souvent confondus par abus de langage.

Le seuillage d'image est une technique simple de binarisation d'image, elle consiste à transformer une image en niveau de gris en une image dont les valeurs de pixels ne peuvent avoir que la valeur 1 ou 0. On parle alors d'une image binaire ou **image en noir et blanc**.

Principe

Le seuillage d'image remplace un à un les pixels d'une image à l'aide d'une valeur seuil fixée (ici `128`). Ainsi, si un pixel a une valeur supérieure ou égale au seuil (par exemple 253), il prendra la valeur `255` (blanc), et si sa valeur est inférieure (par exemple 8), il prendra la valeur `0` (noir).

On va utiliser deux images au format PNG :

Dans la bibliothèque PIL (*Python Imaging Library*), le mode d'une image est une chaîne qui définit le type et la profondeur d'un pixel dans l'image : les différents modes (`'RGB'`, `'P'`, ...).

Felix le chat (PNG (35, 35) RGB → 3 x 8-bits pixels) cette image est "presque binaire"

Dans ce format `'RGB'`, le pixel est codé sur 24 bits soit 3 octets définissant les composantes Rouge Vert Bleu. La valeur de chaque couleur va de `0` à `255`. En Python, la variable pixel sera alors un *tuple* (ici une séquence de 3 entiers). Dans ce cas, on transformera un pixel RGB en niveaux de gris en calculant la moyenne des 3 composantes que l'on conservera sous la forme d'un entier (`int`).

Felix le chat (PNG (220, 228) P → 8-bits pixels)

Dans ce format `'P'`, le pixel est simplement codé sur 8 bits soit 256 niveaux de gris de `0` pour le noir à `255` pour le blanc. En Python, la variable pixel sera tout simplement un entier (`int`).

Tuple en Python

Ce programme manipule des tuples. Les tuples sont des séquences immuables (qui ne peuvent être modifiées), généralement utilisées pour stocker des collections de données hétérogènes.

Dans l'exemple ci-dessus, la variable `p` est un tuple : `p = (255,255,255) # un pixel blanc`

```
# coding: utf-8

print("ACTIVITÉ 2")

# PIL est une bibliothèque de traitement d'image
import PIL;
print("PIL version ", PIL.__version__)

# on importe le module Image de la bibliothèque PIL
from PIL import Image

#"felix-the-cat-nb.png" -> mode RGB
#"felix-le-chat.png"    -> mode P

#import Tkinter as tk # en python 2
import tkinter as tk # en python 3
from tkinter import filedialog

ecran = tk.Tk()
ecran.withdraw()

# choisir un fichier image
fichierImage =  filedialog.askopenfilename(initialdir = "./images/",title = "Choisir
un fichier image",filetypes = (("fichiers png","*.png"), ("fichiers jpeg","*.jpg"
),("fichiers jpeg","*.jpeg"),("all files","*.*")))
if not fichierImage:
    exit()

import os
fichier = os.path.basename(fichierImage)
path = os.path.dirname(fichierImage)
nomImage = os.path.splitext(fichier)[0] # "felix-le-chat"     # mode P
ext = os.path.splitext(fichier)[1] # ".png"

# Une image de felix le chat !
img = Image.open(path + "/" + nomImage + ext);

#imgNG = img.convert('L') # conversion en niveaux de gris
#imgNG.show()

# Affiche les informations sur l'image
```

```
print(img.format, img.size, img.mode)
print("Fichier :", img.filename)
print("L =", img.width, "x H =", img.height)
largeur, hauteur = img.size # Récupération de la largeur et hauteur de l'image
#print("L =", largeur, "x H =", hauteur)
print("Nombre de pixels =", (img.width*img.height) , "pixels")
#print("Nombre de pixels =", (largeur*hauteur) , "pixels")

# Création d'une nouvelle image de même type
nouvelleImage = Image.new(img.mode, img.size)

# Affiche les valeurs des pixels
if(img.width*img.height < 2000): # pour les petites images
    print("Affiche les valeurs des pixels de l'image d'origine")
    for x in range(largeur):
        for y in range(hauteur):
            # récupére les valeurs du pixel
            pixel = img.getpixel((x,y))
            if(isinstance(pixel, int)):
                gris = pixel
            elif(isinstance(pixel, tuple)):
                # calcul du poids de chaque composante du gris dans le pixel (CIE709)
                gris = int(0.2125 * pixel[0] + 0.7154 * pixel[1] + 0.0721 * pixel[2])
                # calcul la valeur de gris par moyenne des 3 canaux RVB
                #gris = int((1/3) * pixel[0] + (1/3) * pixel[1] + (1/3) * pixel[2])
                #gris = int(0.33 * pixel[0] + 0.33 * pixel[1] + 0.33 * pixel[2])
            print(repr(gris).rjust(3), end=' ')
        print("")

# Affiche les pixels en noir (0) ou blanc (1)
if(img.width*img.height < 2000): # pour les petites images
    print("Affiche les pixels en noir (0) ou blanc (1)")

for x in range(largeur):
    for y in range(hauteur):
        # récupére les valeurs du pixel
        pixel = img.getpixel((x,y))
        if(isinstance(pixel, int)):
            gris = pixel
        elif(isinstance(pixel, tuple)):
            # calcul du poids de chaque composante du gris dans le pixel (CIE709)
            gris = int(0.2125 * pixel[0] + 0.7154 * pixel[1] + 0.0721 * pixel[2])
            # calcul la valeur de gris par moyenne des 3 canaux RVB
            #gris = int((1/3) * pixel[0] + (1/3) * pixel[1] + (1/3) * pixel[2])
            #gris = int(0.33 * pixel[0] + 0.33 * pixel[1] + 0.33 * pixel[2])
        # un noir (0) ou un blanc (1) ?
        if(gris > 128):
            if(img.width*img.height < 2000):
                print(repr(1).rjust(1), end=' ') #print("1", end='')
            # un pixel blanc
            if(isinstance(pixel, int)):
```

```
                p = 255
            elif(isinstance(pixel, tuple)):
                p = (255,255,255) # RVB
        else:
            if(img.width*img.height < 2000):
                print(repr(0).rjust(1), end=' ') #print("0", end='')
            # un pixel noir
            if(isinstance(pixel, int)):
                p = 0
            elif(isinstance(pixel, tuple)):
                p = (0,0,0) # RVB
        # ajoute le pixel à la nouvelle image
        nouvelleImage.putpixel((x,y), p)
    if(img.width*img.height < 2000):
        print("")

# Affiche les valeurs des pixels
if(img.width*img.height < 2000): # pour les petites images
    print("Affiche les valeurs des pixels de la nouvelle image")
    for x in range(largeur):
        for y in range(hauteur):
            # récupére les valeurs du pixel
            pixel = nouvelleImage.getpixel((x,y))
            if(isinstance(pixel, int)):
                gris = pixel
            elif(isinstance(pixel, tuple)):
                # calcul du poids de chaque composante du gris dans le pixel (CIE709)
                gris = int(0.2125 * pixel[0] + 0.7154 * pixel[1] + 0.0721 * pixel[2])
                # calcul la valeur de gris par moyenne des 3 canaux RVB
                #gris = int((1/3) * pixel[0] + (1/3) * pixel[1] + (1/3) * pixel[2])
                #gris = int(0.33 * pixel[0] + 0.33 * pixel[1] + 0.33 * pixel[2])
            print(repr(gris).rjust(3), end=' ')
        print("")

# Enregistrement de la nouvelle image
nouvelleImage.save(path + "/" + nomImage + "-nouveau" + ext);

# Affichage de l'image
#img.show()
nouvelleImage.show()

# Fermeture des fichiers
img.close()
nouvelleImage.close()
```

On obtient :

Felix le chat "binarisé"

```
ACTIVITÉ 2
PIL version  5.1.0
PNG (35, 35) RGB
Fichier : ./images/felix-the-cat-nb.png
L = 35 x H = 35
Nombre de pixels = 1225 pixels
Affiche les valeurs des pixels de l'image d'origine
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   8 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   0   0
255 255 255   5 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   4   0   5   0
2 255   6   0 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   4   4   0   0   5
1   0   0   1 255 255 255 255 255 255 255 255 255 255 255
...
Affiche les pixels en noir (0) ou blanc (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 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 1 1 1 0 0 0 0 1 1 1 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 1 1 0 0 0 0 0 0 1 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 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 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 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 0 0 0 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 1 1 1 1 0 0 0 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 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 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 1 1
1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1 1 0 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 1 1 0 1 1 1 1 1 1 1 0 0 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 1 1 0 0 1 1 1 1 1 1 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 0 0 1 1 0 0 0 0 1 1 1 0 0
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 1 0 0 0 0 0 1 1 0 0
1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 0 1 0 1 0 0
1 1 1 1 1 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 1 1 0 0 1 1 0 0 1 0
1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0
1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0
1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0
```

```
1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0
1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 1 0 0 0
1 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 1 0 0 1 1 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 1 1 1 1 1 0 0 0 1 1 0 0 0
1 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 0 0 0 1 0 0 0 1
1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 1
1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1
1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 1
1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1
1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0 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 1 1 1 0 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 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
Affiche les valeurs des pixels de la nouvelle image
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   0   0
255 255 255   0 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   0   0   0
0 255   0   0 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   0   0   0   0   0
0   0   0   0 255 255 255 255 255 255 255 255 255 255
...
```

2.3. Activité n°3 : traitement d'images matricielles

Python n'a pas de type spécifique pour les matrices. On peut utiliser le *package* Python NumPy (https://www.numpy.org/) qui permet de manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques pour le calcul scientifique.

Pré-requis : Les matrices en Python

2.3.1. Matrice transposée

On reprend l'image suivante :

Felix le chat

```python
# coding: utf-8

print("ACTIVITÉ 3a")

# PIL est une bibliothèque de traitement d'image
import PIL;
print("PIL version ", PIL.__version__)
```

```
# on importe le module Image de la bibliothèque PIL
from PIL import Image

import numpy as np
import random as rd

import tkinter as tk # en python 3
from tkinter import filedialog

ecran = tk.Tk()
ecran.withdraw()

# choisir un fichier image
fichierImage =  filedialog.askopenfilename(initialdir = "./images/",title = "Choisir
un fichier image",filetypes = (("fichiers png","*.png"), ("fichiers jpeg","*.jpg"
),("fichiers jpeg","*.jpeg"),("all files","*.*")))
if not fichierImage:
    exit()

import os
fichier = os.path.basename(fichierImage)
path = os.path.dirname(fichierImage)
nomImage = os.path.splitext(fichier)[0] # "felix-le-chat"     # mode P
ext = os.path.splitext(fichier)[1] # ".png"

# Une image de felix le chat !
img = Image.open(path + "/" + nomImage + ext);

# Affiche les informations sur l'image
print(img.format, img.size, img.mode)
print("Fichier :", img.filename)
print("L =", img.width, "x H =", img.height)
largeur, hauteur = img.size # Récupération de la largeur et hauteur de l'image

imgNG = img.convert('L') # conversion en niveaux de gris
imgNG.show()

# image -> tableau numpy (matrice)
matriceImageNG = np.array(imgNG)
lignes, colonnes = matriceImageNG.shape
print(lignes, "x", colonnes)
print(matriceImageNG.ndim, "dimensions")
print("")

# Affiche les pixels en noir (0) ou blanc (1)
#print(matriceImageNG)
for i in range(0,lignes):
    for j in range(0,colonnes):
        if(matriceImageNG[i,j] > 128):
            print(repr(1).rjust(1), end=' ')
        else:
```

```
            print(repr(0).rjust(1), end=' ')
        # ou :
        #print(repr(matriceImageNG[i,j]).rjust(3), end=' ')
    print("")
print("")

# transposée de l'image imgNG avec numpy
#matriceTransposeImageNG = matriceImageNG.transpose()
#print(matriceTransposeImageNG)

# transposée de l'image imgNG à la main
matriceTransposeImageNG = matriceImageNG.copy()
for i in range(0,lignes):
    for j in range(0,colonnes):
       matriceTransposeImageNG[j][i] = matriceImageNG[i][j]

# Affiche les pixels noir (0) ou blanc (1)
#print(matriceTransposeImageNG)
for i in range(0,lignes):
    for j in range(0,colonnes):
        if(matriceTransposeImageNG[i,j] > 128):
            print(repr(1).rjust(1), end=' ')
        else:
            print(repr(0).rjust(1), end=' ')
    print("")
print("")

# tableau numpy (matrice) -> image
imageNG = Image.fromarray(matriceTransposeImageNG, 'L')
imageNG.show()

imageNG.save(path + "/" + nomImage + "-transpose" + ext);

# Autres : image aléatoire
#matriceAleatoire = np.array([[rd.randrange(256) for k in range(100)] for i in
range(100)])
#imageAleatoire = Image.fromarray(matriceAleatoire, 'L')
#imageAleatoire.show()

# Fermeture des fichiers
img.close()
imageNG.close()
```

On obtient :

Felix le chat "transposé"

```
ACTIVITÉ 3a
```

```
PIL version  5.1.0
PNG (35, 35) RGB
Fichier : ./images/felix-the-cat-nb-nouveau.png
L = 35 x H = 35
35 x 35
2 dimensions

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

2.3.2. Symétrie et rotation

D’autres opérations peuvent être réalisées :

Symétrie horizontale

Symétrie verticale

Rotation

```
# Symétrie horizontale
symetrieHorizontaleImageNG = matriceImageNG.copy()
for x in range(0,colonnes):
    for y in range(0,lignes):
       symetrieHorizontaleImageNG[-x+colonnes-1][y] = matriceImageNG[x][y]

imageHorizontaleNG = Image.fromarray(symetrieHorizontaleImageNG, 'L')
imageHorizontaleNG.show()
imageHorizontaleNG.save(path + "/" + nomImage + "-symetrie-horizontale" + ext);

# Rotation
rotationImageNG = matriceImageNG.copy()
for x in range(0,colonnes):
    for y in range(0,lignes):
       rotationImageNG[colonnes-x-1][lignes-y-1] = matriceImageNG[x][y]

imageRotationNG = Image.fromarray(rotationImageNG, 'L')
imageRotationNG.show()
imageRotationNG.save(path + "/" + nomImage + "-rotation" + ext);
```

2.3.3. Filtres de couleur

Un filtre coloré transmet ("laisse passer") certaines lumières colorées et absorbe les autres.

La couleur d'un filtre est déterminée par la ou les lumières colorée(s) qu'il transmet. Un filtre de couleur primaire ne transmet que la lumière de cette couleur (rouge jaune bleu) :

- un filtre jaune transmet les lumières verte et rouge et absorbe la lumière bleue.
- un filtre magenta transmet les lumières bleue et rouge et absorbe la lumière verte.
- un filtre cyan (bleu) transmet les lumières verte et bleue et absorbe la lumière rouge.

La synthèse soustractive des couleurs est le procédé consistant à combiner l'absorption d'au moins trois colorants pour obtenir les nuances d'une gamme.

On emploie l'adjectif soustractive par opposition à la synthèse additive. Ce terme est cependant trompeur, car les primaires n'effectuent pas, sur la lumière de l'éclairant, une soustraction, mais une multiplication, différente pour chaque partie du spectre, par un nombre compris entre 0 et 1.

```
# coding: utf-8

print("ACTIVITÉ 3c")

# PIL est une bibliothèque de traitement d'image
import PIL;
print("PIL version ", PIL.__version__)
```

```
# on importe le module Image de la bibliothèque PIL
from PIL import Image

import numpy as np

import tkinter as tk # en python 3
from tkinter import filedialog

ecran = tk.Tk()
ecran.withdraw()

# choisir un fichier image
fichierImage =  filedialog.askopenfilename(initialdir = "./images/",title = "Choisir
un fichier image",filetypes = (("fichiers png","*.png"), ("fichiers jpeg","*.jpg"
),("fichiers jpeg","*.jpeg"),("all files","*.*")))
if not fichierImage:
    exit()

import os
fichier = os.path.basename(fichierImage)
path = os.path.dirname(fichierImage)
nomImage = os.path.splitext(fichier)[0] # "felix-le-chat"     # mode P
ext = os.path.splitext(fichier)[1] # ".png"

# Une image de felix le chat !
img = Image.open(path + "/" + nomImage + ext);

# Affiche les informations sur l'image
print(img.format, img.size, img.mode)
print("Fichier :", img.filename)
print("L =", img.width, "x H =", img.height)
largeur, hauteur = img.size # Récupération de la largeur et hauteur de l'image

#img.show()

# Redimensionnement de l'image source en une image de taille moitié
imageVignette = img.resize((largeur // 2, hauteur // 2), Image.BICUBIC) # Division par
2 de la taille (// permet de ne récupérer que le quotient de la division)
largeurVignette,hauteurVignette= imageVignette.size

# Création des vignettes avec filtre de couleur
image1 = Image.new(imageVignette.mode,imageVignette.size) #jaune R255 V255 B0
image2 = Image.new(imageVignette.mode,imageVignette.size)
image3 = Image.new(imageVignette.mode,imageVignette.size)
image4 = Image.new(imageVignette.mode,imageVignette.size)

# boucle de traitement des pixels pour appliquer filtre couleur
for x in range(largeurVignette):
    for y in range(hauteurVignette):
        pixel = imageVignette.getpixel((x,y))
        # Filtre jaune : absorbe le bleu
```

```
        #p = (int(255*0.4 + pixel[0]*0.6), int(255*0.4 + pixel[1]*0.6), int(0*0.4 +
pixel[2]*0.6))
        p = (int(pixel[0]), int(pixel[1]), int(0))
        image1.putpixel((x,y), p)
        # Filtre magenta : absorbe le vert
        #p = (int(255*0.4 + pixel[0]*0.6), int(0*0.4 + pixel[1]*0.6), int(255*0.4 +
pixel[2]*0.6))
        p = (int(pixel[0]), int(0), int(pixel[2]))
        image2.putpixel((x,y), p)
        # Filtre bleu : absorbe le rouge
        #p = (int(0*0.4 + pixel[0]*0.6), int(255*0.4 + pixel[1]*0.6), int(255*0.4 +
pixel[2]*0.6))
        p = (int(0), int(pixel[1]), int(pixel[2]))
        image3.putpixel((x,y), p)
        # Filtre vert : absorbe le rouge et le bleu
        #p = (int(0*0.4 + pixel[0]*0.6), int(255*0.4 + pixel[1]*0.6), int(0*0.4 +
pixel[2]*0.6))
        p = (int(0), int(pixel[1]), int(0))
        image4.putpixel((x,y), p)

# Création d'une image avec les 4 vignettes colorées
imageFinale = Image.new(img.mode,img.size)
imageFinale.paste(im=image1, box=(0, 0))
imageFinale.paste(im=image2, box=(largeurVignette, 0))
imageFinale.paste(im=image3, box=(0, largeurVignette))
imageFinale.paste(im=image4, box=(largeurVignette, largeurVignette))

# Affichage de l'image et enregistrement
imageFinale.show()
imageFinale.save("./images/" + nomImage + "-filtres-couleurs" + ext)

# Fermeture du fichier image
img.close()
```

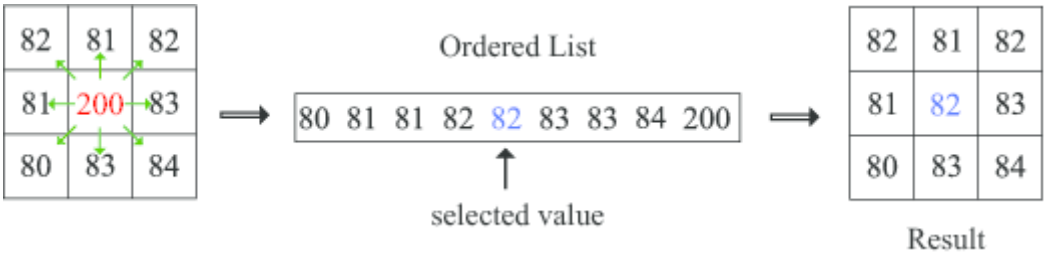
Image "Pop art" obtenue

2.3.4. Filtrage matriciel : le filtre médian

Un filtre médian est très efficace pour enlever un bruit impulsif (sous forme de parasites isolés) qui peuvent avoir une amplitude très grande. Il permet de supprimer le bruit sans altérer les détails de l'image.

L'idée principale du filtre médian est de remplacer chaque entrée par la valeur médiane de son voisinage.

Extrait de l'article Matrices et Images Numériques d'Antoine Nectoux



Le filtre médian permet d'éliminer les valeurs aberrantes (ici la valeur `200`) sans se limiter à faire un calcul de moyenne qui aura tendance à contaminer les valeurs voisines avec cette valeur aberrante et flouter l'image.

Les images de Lenna avec du bruit : http://agamenon.tsc.uah.es/Investigacion/gram/papers/Noise/Images/lenna/

Lenna avec du "bruit"

```
# coding: utf-8

print("ACTIVITÉ 3d")

# PIL est une bibliothèque de traitement d'image
import PIL;
print("PIL version ", PIL.__version__)

# on importe le module Image de la bibliothèque PIL
from PIL import Image, ImageFilter

import numpy as np

# Lenna !
#nomImage = "lenna_10"
#ext = ".jpeg"

import tkinter as tk # en python 3
from tkinter import filedialog

ecran = tk.Tk()
ecran.withdraw()
```

```
# choisir un fichier image
fichierImage =  filedialog.askopenfilename(initialdir = "./images/",title = "Choisir
un fichier image",filetypes = (("fichiers jpeg","*.jpg"),("fichiers jpeg","*.jpeg"
),("fichiers png","*.png"),("all files","*.*")))
if not fichierImage:
    exit()

import os
fichier = os.path.basename(fichierImage)
path = os.path.dirname(fichierImage)
nomImage = os.path.splitext(fichier)[0] # "felix-le-chat"     # mode P
ext = os.path.splitext(fichier)[1] # ".png"

# Une image avec du bruit
img = Image.open(path + "/" + nomImage + ext);

# Affiche les informations sur l'image
print(img.format, img.size, img.mode)
print("Fichier :", img.filename)
print("L =", img.width, "x H =", img.height)
largeur, hauteur = img.size

imgNG = img.convert('L') # conversion en niveaux de gris
imgNG.show()

# image -> tableau numpy
matriceImageNG = np.array(imgNG)
lignes, colonnes = matriceImageNG.shape
print(lignes, "x", colonnes)
print(matriceImageNG.ndim, "dimensions")

if lignes != colonnes:
    exit()

# Principe : Filtre médian (3 x 3)
filtreMedianImageNG = matriceImageNG.copy()
matricePixels = np.zeros((9))

for i in range(colonnes-1):
    for j in range(lignes-1):
        if j > 0 and i > 0:
            matricePixels[0] = matriceImageNG[i-1][j-1]
            matricePixels[1] = matriceImageNG[i-1][j]
            matricePixels[2] = matriceImageNG[i-1][j+1]
            matricePixels[3] = matriceImageNG[i][j-1]
            matricePixels[4] = matriceImageNG[i][j]
            matricePixels[5] = matriceImageNG[i][j+1]
            matricePixels[6] = matriceImageNG[i+1][j-1]
            matricePixels[7] = matriceImageNG[i+1][j]
            matricePixels[8] = matriceImageNG[i+1][j+1]
            s = np.sort(matricePixels, axis=None)
```

```
            filtreMedianImageNG[i][j] = s[4] # on prend la valeur médiane

filtreMedianNG = Image.fromarray(filtreMedianImageNG, 'L')
filtreMedianNG.show()
filtreMedianNG.save(path + "/" + nomImage + "-filtre-median" + ext);

# sinon avec le module ImageFilter (5 x 5)
im2 = imgNG.filter(ImageFilter.MedianFilter(5))
im2.show()

# Fermeture des fichiers
img.close()
filtreMedianNG.close()
```

Lenna avec un filtre médian de réduction du bruit

2.3.5. Bibliothèque PIL (*Python Imaging Library*)

Les opérations précédentes peuvent être réalisées directement avec la bibliothèque PIL (*Python Imaging Library*) !

La fonction `transpose()` permet de retourner ou faire pivoter par incréments de 90 degrés une image :

```
from PIL import Image

im = Image.open("image.jpg")

# Retourner l'image de gauche à droite
im_flipped = im.transpose(method=Image.FLIP_LEFT_RIGHT)

# autres : Image.FLIP_TOP_BOTTOM, Image.ROTATE_90, Image.ROTATE_180, Image.ROTATE_270,
Image.TRANSPOSE ou Image.TRANSVERSE
```

D'autre part, le module ImageFilter fournit un ensemble suivant de filtres d'amélioration d'image prédéfinis;

La méthode `Image.filter()` :

```
from PIL import Image, ImageFilter

im = Image.open("image.jpg")

im1 = im.filter(ImageFilter.BLUR)

im_blurred = im.filter(ImageFilter.GaussianBlur(radius=5))
im_blurred.show()

# Filtre médian
im2 = im.filter(ImageFilter.MedianFilter(3)) # 3 x 3
im2.show()

# autres : CONTOUR, DETAIL, EDGE_ENHANCE, EDGE_ENHANCE_MORE, EMBOSS, FIND_EDGES,
SHARPEN, SMOOTH et SMOOTH_MORE
```

2.3.6. Voir aussi

- Les filtres par convolution, passe-haut et passe-bas et la détection de contours : Filtrage et traitement d'images (TangenteX.com)
- La détection de Canny Edge et les filtres gaussien et Sobel : ichi.pro/fr/

Codes sources des exemples : https://github.com/tvaira/python-traitement-images.

Site : tvaira.free.fr