Traitement d'images (Partie 1)

Objectifs

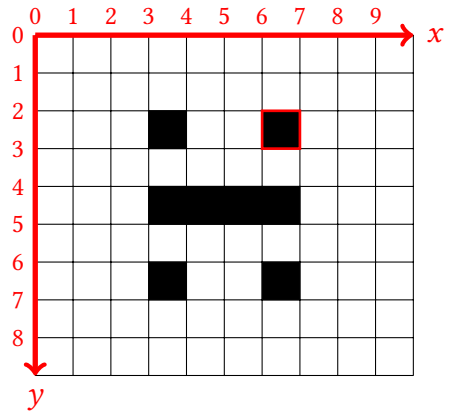
- connaître la représentation en mémoire d'une image numérique
- savoir ouvrir, retailler, afficher et sauvegarder une image
- savoir manipuler une image au niveau du pixel
- approfondir la notion de programmation orienté objet
- apprendre à documenter son code

Pour manipuler une image en Python, il faut utiliser une bibliothèque spécialisée. Dans ce TP, nous utiliserons le module *Pillow* dont la documentation se trouve à l'adresse suivante : https://pillow.readthedocs.io/en/stable/
Pensez à vous y reporter régulièrement. C'est une source d'informations importantes qui vous expose les méthodes disponibles et les possibilités offertes par le module.

Connaissances et savoir-faire à maîtriser

1) Encodage d'une image

Une image numérique est encodé sous la forme d'un tableau à deux dimensions dans lequel chaque élément est appelé **un pixel** (pour *PICture ELement*). On accède à un pixel de cette matrice par ses coordonnées définies dans un repère dont l'origine est le coin supérieur gauche de l'image.



Dans l'exemple ci-contre, les dimensions de l'image sont de 10 pixels de large par 9 pixels de haut : l'image contient donc 90 pixels.
Le pixel encadré est celui de coordonnées (6 ; 2) *: comme pour les listes et les tuples, les index commencent à zéro.*

2) Espace colorimétrique d'une image

Une image possède un attribut appelé `mode` qui définit son espace colorimétrique (on parle aussi de **profondeur de couleur**), autrement dit, la façon dont la couleur est encodée au niveau de chaque pixel. On crée une nouvelle image via l'instruction :

```
img = Image.new(mode, (largeur, hauteur)) # mode = "1", "L" ou "RGB"
```

a. Le mode `1` noir et blanc

Dans ce mode la couleur de chaque pixel est codé sur 1 bit (*8 bits = 1 octet*) qui vaut soit 0 (noir), soit 1 (blanc). On affecte une couleur au pixel avec la méthode `putpixel` où `(x, y)` représente les coordonnées du pixel à coloriser.

```
img.putpixel((x, y), 0) # pour du noir
img.putpixel((x, y), 1) # pour du blanc
```

b. Le mode `L` niveaux de gris

La couleur prend toutes les valeurs entières qu'il est possible d'encoder sur un octet, soit 256 valeurs qui vont du noir "total" (0) au blanc "pur" (255) en passant par tous les niveaux de gris :

```
img.putpixel((x, y), 127) # pour du gris "médian"
```

Remarque 1 *: il existe par conséquent 256 niveaux de gris différents.*

c. Le mode couleur `RGB`

La couleur est ici encodée sur trois octets soit $3 \times 8 = 24$ bits (un octet par couleur, on peut dire aussi par **canal**). Chaque "teinte" de rouge, de vert ou de bleu prend une valeur entière entre 0 et 255. La couleur finale est la synthèse additive des trois canaux.

```
img.putpixel((x, y), (65, 147, 221)) # 3-tuple (r, g, b) codant pour du bleu ciel
```

Remarque 2 *: on peut ainsi encoder plus de 16 millions de couleurs* ($256^3 = 16\,777\,216$).

Remarque 3 *: il existe d'autres modes comme par exemple le mode* `RGBA` *dans lequel un pixel est encodé avec un octet supplémentaire, ce qui permet de gérer un effet de transparence (*`A` *pour* ***canal alpha****).*

d. En résumé

Mode	Pixel sur …	Couleurs disponibles	Poids en mémoire en octets
1	1 bit	2 (noir ou blanc)	$N/8$
L	8 bits = 1 octet	256 niveaux de gris	N
RGB	24 bits = 3 octets	plus de 16 millions	$N \times 3$

(N = nombre de pixels de l'image)

3) Ouvrir, manipuler et sauver une image

Les manipulations au niveau du pixel se font à travers les deux méthodes d'images `getpixel` et `putpixel`. La première permet de récupérer le code RGB d'un pixel en indiquant ses coordonnées en paramètre sous forme d'un 2-tuple `(x, y)`. La deuxième modifie la couleur d'un pixel, elle prend les coordonnées `(x, y)` du pixel à modifier et le code `(r, g, b)` à lui attribuer. Dans l'exemple suivant, on modifie le pixel en ne gardant que sa composante de rouge, on dit qu'*on filtre le rouge.*

```
from PIL import Image # importation du module
img = Image.open("perroquet.jpg") # ouvre une image
(r, g, b) = img.getpixel((20, 35)) # récupère le code RGB du pixel dans un tuple
img.putpixel((20, 35), (r, 0, 0)) # change la couleur du pixel
img.show() # affiche l'image sans l'enregistrer dans l'editeur par défaut
```

4) Retailler une image

La méthode d'image `resize` permet de changer les dimensions d'une image. Il faudra toutefois veiller à conserver ses proportions (on parle de **ratio**) afin d'éviter toute déformation :

```
img.resize((int(img.width / 2), int(img.height / 2))) # échelle 1/2
img.save("perroquet.png", "PNG") # enregistre l'image au format PNG
```

5) La classe `Image` en détail

On a déjà vu dans un pécédent TP qu'un langage comme Python manipule les données sous forme d'objets. Une chaîne de caractère par exemple est un objet de type `string` qui possède des méthodes permettant d'agir sur celui-ci. Un objet possède aussi des **attributs** : ce sont des propriétés qui décrivent l'état de cet objet. Deux objets `Image` différents vont avoir des propriétés différentes mais vont être "construits" suivant le même modèle. Ce modèle qui sert à créer un objet s'appelle une **classe d'objet**. Ci-dessous une brève description de la classe `Image` :

```
+ ------------------------------------------------ +
| Classe Image                                     |
+ ------------------------------------------------ +
| ATTRIBUTS :                                      |
|     |-- filename : nom de l'image                |
|     |-- format   : son encodage (jpg, png, ...)  |
|     |-- mode     : mode couleur (1, L, RGB, ...) |
|     |-- width    : largeur de l'image            |
|     |-- height   : hauteur de l'image            |
+ ------------------------------------------------ +
| METHODES :                                       |
|     |-- open(...)                                |
|     |-- new(...)                                 |
|     |-- save(...)                                |
|     |-- getpixel(...)                            |
|     |-- putpixel(...)                            |
|     |-- show(...)                                |
|     |-- resize(...)                              |
+ ------------------------------------------------ +
```

Pour une description complète des méthodes et attributs de la classe `Image`, reportez vous à la doc https://pillow.readthedocs.io/en/stable/reference/Image.html.

Remarque 4 *: on retiendra à ce stade que la* ***classe*** *est une notion abstraite qui décrit un objet et que l'objet en lui-même est un exemplaire de cette classe, on dit que l'objet est* ***une instance de classe****.*

6) Parcourir les pixels d'une image

Pour parcourir l'ensemble des pixels d'une image, on utilise la technique de la double boucle FOR imbriquée :

```
for y in range(img.height): # parcours ligne par ligne
    for x in range(img.width): # et colonne par colonne
        img.putpixel((x, y), (127, 127, 127)) # tout en gris
```

Exercices à réaliser

Afin d'acquérir de bonnes pratiques de programmation, nous allons prendre l'habitude d'ajouter sous forme de commentaires des informations sur les fonctions que l'on écrit. On dit que l'on **documente son code**. Vous êtes libre d'adopter le style qui vous convient, par exemple :

```
def printRGB(pimg, ppixel):
    """
    Description : affiche le code RGB de ppixel dans l'image pimg
    Paramètres  : type(pimg) => Image.PIL
                  type(ppixel) => 2-tuple : (x, y)
    Retour      : aucun
    """
```

Il est d'usage également de créer une en-tête dans son fichier source indiquant quelques informations importantes :

```
"""
Programme   : TP-images-exo1.py
Langage     : Python 3.7.3
Modules     : Pillow 7.1.2
Auteur      : Mathieu Pons
Description : introduction à la manipulation d'images
Docs Pillow : https://pillow.readthedocs.io/en/stable/
"""
```

Exercice 1

Télécharger l'image `perroquet.jpg` **dans le même répertoire** que le fichier `TP-images-exo1.py` et répondre aux questions suivantes :

1. Écrire une fonction `printRGB(pimg, ppixel)` tel que définie précédemment et la tester.
2. Écrire une fonction `afficher_info(pimg)` qui reproduit l'affichage suivant :

```
+ ------------------------------------------------------ +
| perroquet.png                                          |
+ ------------------------------------------------------ +
| FORMAT                          : PNG                  |
| MODE                            : Couleurs RGB         |
| TAILLE                          : 960 x 640            |
| NB PIXELS                       : 614400               |
| POIDS(non compressé) en Mo      : 1.76                 |
+ ------------------------------------------------------ +
```

   Le poids non compressé d'une image correspond à son occupation en mémoire de travail qui est forcément plus importante que son poids sur disque sur lequel elle est stockée sous forme compressée (format JPG ou PNG ...). Attention, 1 *Ko* = 1 024 *octets et* 1 *Mo* = 1 024 *Ko.*

3. Écrire une fonction `retailler_sauver(pimg, plarg)` qui modifie la largeur de l'image en conservant ses proportions et l'enregistre au format PNG :

```
def retailler_sauver(pimg, plarg):
    """
    Description : modifie la largeur de l'image en conservant son ratio
                  et l'enregistre au format PNG
    Paramètres  : type(pimg) => Image.PIL
                  type(plarg) => int
    Retour      : Image.PIL
    """
```

Exercice 2

Pour cet exercice, vous pouvez continuez à travailler sur le fichier `perroquet.jpg` ou une autre image de votre choix. Ouvrir le fichier `TP-images-exo2.py` et répondre aux questions suivantes :

1. Écrire une fonction `convertir_NB(pimg)` qui convertit l'image passée en paramètre en noir et blanc. Votre fonction devra se charger de créer une nouvelle image vide en mode `1` de même taille que l'image à convertir :

```
def convertir_NB(pimg):
    """
    Description : convertit pimg en noir et blanc
    Paramètres  : type(pimg) => Image.PIL
    Retour      : Image.PIL
    """
```

2. Écrire une fonction `convertir_gris(pimg)` qui convertit l'image passé en paramètre en niveaux de gris sur le même principe que la question précédente.
3. Écrire une fonction `filtrer_canaux(pimg)` qui filtre chaque canal de rouge, de vert et de bleu de l'image passée en paramètre et renvoit un 3-tuple contenant les images filtrées :

```
def filtrer_canaux(pimg):
    """
    Description : filtre le rouge, le vert et le bleu de pimg
    Paramètres  : type(pimg) => Image.PIL
    Retour      : 3-tuple : (Image.PIL, Image.PIL, Image.PIL)
    """
```

Remarques importantes :

- pour convertir en noir et blanc, l'idée est de faire la moyenne des trois composantes d'un pixel et de la comparer à un seuil que vous fixez vous-même. Si cette moyenne est au-dessus du seuil, on passe le pixel en blanc sinon on le passe en noir ;
- pour la conversion en niveau de gris, on calcule la *luminance* L du pixel selon la formule :

$$L = 0.2126 * R + 0.7152 * G + 0.0722 * B$$

  et on l'affecte au pixel en veillant au préalable à transtyper la valeur de L en un entier ;
- filtrer un canal de couleur revient à ne garder dans le code RGB que la couleur à filtrer. Par exemple, si on filtre le rouge, on garde la composante de rouge inchangée dans le code RGB et on fixe les deux autres à 0.

Les images obtenues avec les filtres précédents sont les suivantes :

Figure 1 – Noir et blanc

Figure 2 – Original

Figure 3 – Niveaux de gris

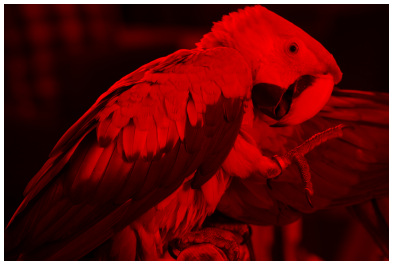


Figure 4 – Filtre rouge

Figure 5 – Filtre vert

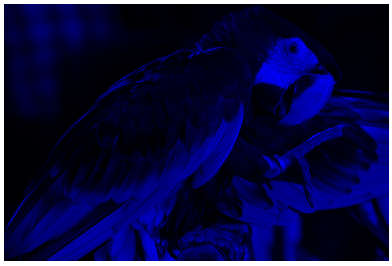


Figure 6 – Filtre bleu

Compétences à valider

Traitement d'images		☼	⛅	🌧	⛈
Théorie	Je sais comment est encodée une image sous forme numérique				
	J'ai compris comment est encodée la couleur d'un pixel en fonction du mode de couleur d'une image				
	Je sais calculer le poids "réel" d'une image lorsque celle-ci est stockée en mémoire de travail				
Pratique	Je sais ouvrir une image et accéder à ses pixels				
	Je sais créer une image vierge dans le mode approprié				
	Je sais afficher, retailler et sauvegarder une image dans le format de mon choix				
	Je sais parcourir une image et modifier la couleur de chaque pixel				