

TP2

Manipulation d'images avec Pillow



1 – Présentation du TP

Objet du TP

Ce TP consiste à apprendre les bases du traitement d'images avec le module Python « Pillow ».

Créez un projet « tp2 » dans lequel vous ajouterez un fichier Python nommé « tp2.py ».

Compétences travaillées dans ce TP

- Manipulations de base sur une image avec Pillow

2 – Installation de Pillow

Qu'est-ce que Pillow ?

PIL (Python Image Library) est une des bibliothèques de référence en Python pour le traitement d'images, supportée jusqu'à Python 2.7.

Le module Pillow reprend et complète les fonctionnalités d'origine de PIL en Python 3. C'est ce module que nous allons utiliser dans ce TP.



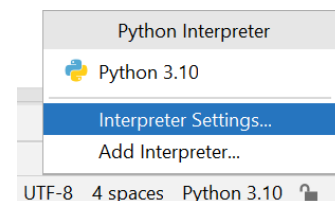
Ne faisant pas partie des modules de base de Python, il faut l'installer avec le gestionnaire de paquets pip.

Installation avec PyCharm

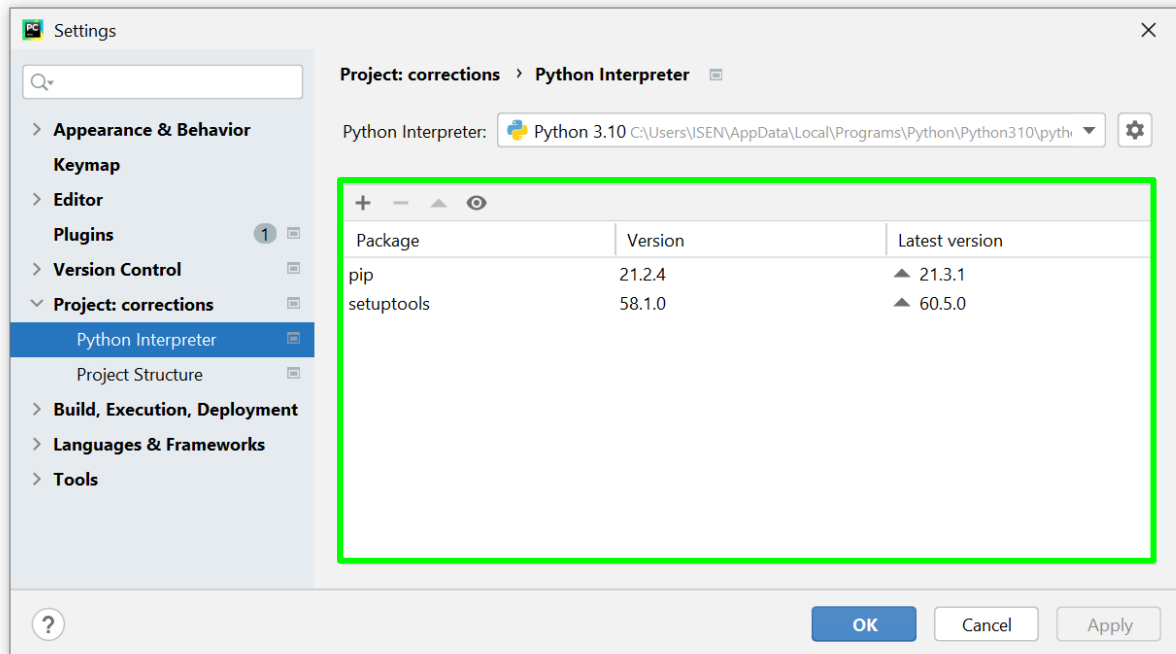
Comme nous utilisons PyCharm, nous allons pouvoir effectuer cette installation en mode « clic-clic ». C'est l'IDE qui se chargera de générer et d'exécuter les commandes d'installation pour nous, telles que `pip install Pillow`.

Pour installer Pillow sous PyCharm, suivez les instructions suivantes :


- En bas à droite de la fenêtre, vous devriez voir un texte de la forme « Python x.y », qui correspond à l'interpréteur Python utilisé dans votre projet (ici Python 3.10). Cliquez dessus, puis sur « Interpreter Settings » dans le menu.



- Vous deviez voir une fenêtre de la forme suivante s'ouvrir :



La partie encadrée en vert liste les paquets Python déjà installés, et leur version. Quand vous venez d'installer Python, cette liste est quasiment vide.

Pour installer un nouveau paquet, cliquez sur .

- Une nouvelle fenêtre apparaît. Dans le champ de recherche, tapez « Pillow », sélectionnez le premier élément de la liste appelé « Pillow » puis cliquez sur « Install package ».
- Dans la liste, « Pillow » va être renommé en « Pillow (installing) » pour indiquer que l'installation du paquet est en cours. Cette installation peut prendre quelques dizaines de secondes, soyez patients ! Au bout d'un moment, vous allez voir un message dans le bas de la fenêtre indiquant que l'installation est terminée :

Package 'Pillow' installed successfully

- Fermez la fenêtre de recherche.
- Pillow devrait maintenant se trouver dans la liste des paquets installés. Cliquez sur OK, c'est terminé !



Il est à noter que cette procédure est valable pour n'importe quel paquet. Il vous sera demandé de la reproduire dans des TP ultérieurs pour installer d'autres paquets.

Vérification de l'installation

Pour vérifier que Pillow a été installé correctement, ajoutez la ligne suivante à votre code :

```
from PIL import Image
```

S'il y a une erreur, il y a eu un problème avec votre installation.



Vous noterez que le module importé s'appelle « PIL » et non « Pillow », de manière à ce que Pillow soit compatible avec son ancêtre PIL.

3 – Modélisation des images dans ce TP

Avant de commencer, nous allons voir le format utilisé pour représenter les images manipulées dans ce TP.

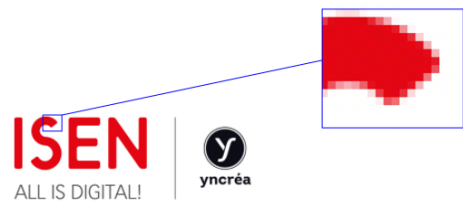
Les images matricielles

Il existe deux grands types d'images : les images *matricielles* et les images *vectorielles*.

Une image vectorielle est constituée de formes géométriques élémentaires (segments de droite, arcs de cercles, courbes de Bézier...).

Une image matricielle est constituée de petits carrés colorés appelés *pixels* (picture elements).

L'exemple ci-contre met en évidence les pixels du logo ISEN.



Le modèle RGB

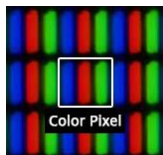
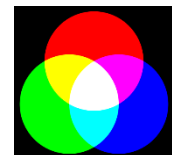
Pour coder la valeur d'un pixel, on utilise 3 composantes.

Dans le modèle RGB, les composantes sont 3 couleurs primaires :

- R = rouge (*red*)
- G = vert (*green*)
- B = bleu (*blue*)

Ces trois composantes sont combinées par synthèse additive (illustrée ci-contre)

- Toutes composantes au maxi → blanc
- Toutes composantes au mini → noir



C'est ce procédé qui est utilisé dans les écrans ou dans les projecteurs (pixel d'un écran LCD ci-contre). A noter que ce n'est pas le cas des imprimantes, qui procèdent par synthèse soustractive.

D'autres modèles existent, comme HSV (Teinte, Saturation, Valeur), que l'on retrouve notamment dans les logiciels de retouche d'image.

Modélisation des pixels

















Dans les formats matriciels, les composantes RGB sont habituellement stockées sous la forme de 3 entiers, chaque entier prenant une valeur entre 0 et 255.

Quelques exemples de couleurs sont illustrés ci-contre. Il est à noter que les nuances de gris ont des composantes RGB égales.

La bibliothèque que nous allons utiliser dans ce TP, Pillow, prend en charge plusieurs modèles.

Si on utilise le modèle RGB, alors un pixel sera modélisé sous la forme d'un tuple (R,G,B), chaque composante étant un entier compris entre 0 et 255. Par exemple, un pixel rouge sera modélisé par le tuple (255, 0, 0).

Comme les tuples sont immuables, vous ne pourrez pas modifier les pixels d'une image. Si vous voulez appliquer des traitements sur des pixels, il vous faudra en créer de nouveaux.

	R	G	B
	0	0	0
	255	255	255
	224	224	224
	128	128	128
	64	64	64
	255	0	0
	255	96	208
	160	32	255
	80	208	255
	0	32	255
	96	255	128
	0	192	0
	255	224	32
	255	160	16
	160	128	96
	255	208	160

4 – Prise en main

Avant de réaliser des traitements sur des images, les exercices suivants vont vous permettre de vous familiariser avec les opérations de base de Pillow.

Écran bleu

Commençons par voir comment créer et visualiser une image vierge.

Tout d'abord, l'appel :

```
Image.new(mode, size, color)
```

permet de créer une image ayant le modèle de couleurs `mode`, la taille `size`, et la couleur de fond `color` :

- `mode` est une chaîne de caractères. Dans ce TP, nous allons utiliser le modèle RGB, modélisé par la chaîne "RGB"
- `size` un tuple de la forme (largeur, hauteur) exprimé en pixels. **Attention de ne pas inverser ces deux dimensions !**
- `color` est un tuple (r, g, b), expression de la couleur de fond dans le modèle RGB. Chaque composante de la couleur est un entier allant de 0 à 255.

Ensuite, étant donnée une image `img`, l'appel :

```
img.show()
```

permet de l'afficher avec le visualiseur par défaut sur votre système.

Ainsi, le code suivant affiche une image bleue de dimensions 450x300 :

```
from PIL import Image

fond_bleu = Image.new("RGB", (450, 300), (0, 0, 255))
fond_bleu.show()
```

Recopiez ce code et lancez votre programme pour vérifier qu'il produit le résultat souhaité.

Note : dans toute la suite du TP, on ne réécrira pas la ligne correspondant à l'import du module PIL, que l'on considérera présente au début de votre code.

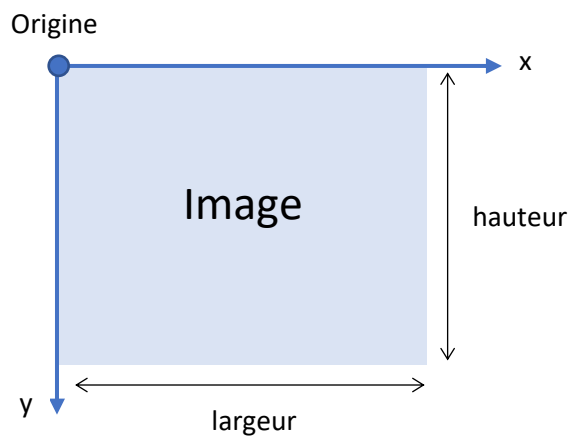
Drapeau français

Étant donnée une image `img`, l'appel :

```
img.putpixel(coords, pixel)
```

affecte la valeur du pixel situé aux coordonnées `coords` à `pixel` :

- `pixel` est un tuple `(r, g, b)` de la même forme que pour la couleur de fond dans `Image.new()`
- `coords` est un tuple `(x,y)` dans le système de coordonnées suivant (exprimé en pixels) :

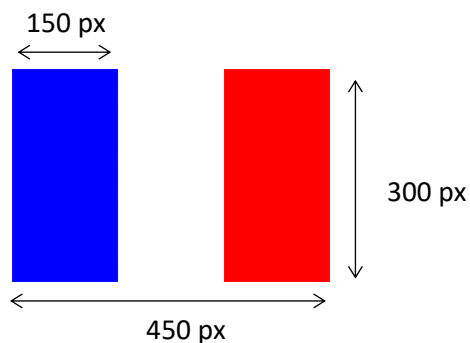


Ce système sera utilisé par toutes les fonctions de Pillow manipulant des coordonnées.

En utilisant ces informations, implémentez une fonction :

```
def creer_drapeau_francais() :
```

qui crée et renvoie une image de taille représentant le drapeau français, conformément aux spécifications suivantes :



Attention : la fonction doit **créer** une image, et non **pas l'afficher**. Pour tester votre fonction, il faut afficher le retour de votre fonction comme ceci :

```
drapeau_fr = creer_drapeau_francais()
drapeau_fr.show()
```

Dégradé de gris

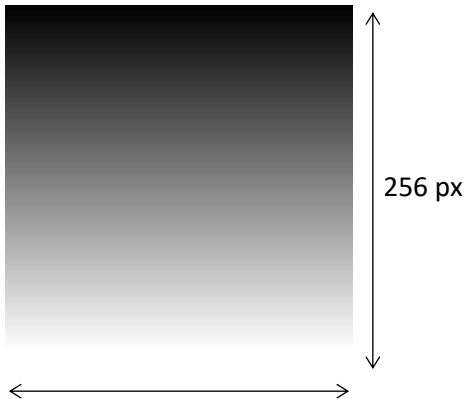
Nous avons vu en cours que les niveaux de gris avaient des composantes r, g, b égales.

En utilisant cette information, implémentez une fonction :

```
def creer_degrade_vertical() :
```

qui crée et renvoie un dégradé de gris vertical de taille 256 x 256 pixels, tel qu'illustré dans l'exemple d'utilisation ci-après.

Exemple d'utilisation :

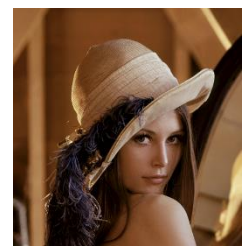
Code	Résultat
<pre>degrade = creer_degrade_vertical() degrade.show()</pre>	

5 – Manipulations de base sur une image

Image de référence

L'image ci-contre, appelée *Lena*, est une image de référence dans la communauté du traitement d'images.

Récupérez le fichier « lena.png » sur la page Moodle du cours et placez-là **dans le répertoire de votre projet**.



Chargement d'une image

L'appel :

```
image = Image.open(nom_fichier)
```

crée une image à partir du contenu lu dans le fichier dont le nom est spécifié en argument.

Exécutez le code suivant et vérifiez qu'il affiche bien Lena :

```
lena = Image.open("lena.png")  
lena.show()
```

Dimensions d'une l'image

Étant donnée une image `img`, `img.size` contient les dimensions de l'image, sous la forme d'un tuple (largeur, hauteur). **Attention à ne pas inverser ces dimensions !**

Par exemple, le code suivant affiche les dimensions de Lena :

```
print(lena.size) # affiche (456, 490)
```

Ce qui signifie que Lena fait 456 pixels de large et 490 de haut.

Astuce

On peut récupérer les éléments d'un tuple dans plusieurs variables en une seule affectation.

Par exemple, la ligne suivante permet de récupérer la largeur et la hauteur de Lena dans deux variables séparées :

```
largeur, hauteur = lena.size
```

Ceci évite d'écrire l'équivalent en deux lignes :

```
largeur = lena.size[0]  
hauteur = lena.size[1]
```

Lecture des pixels d'une image

Étant donnée une image `img`, l'appel :

```
img.getpixel(coords)
```

renvoie, sous la forme d'un tuple (r,g,b), le pixel situé aux coordonnées `coords` (tuple de la forme (x,y), les coordonnées étant exprimées dans le repère présenté dans la partie 3).

Testons ceci sur Lena. La ligne de code :

```
pixel = lena.getpixel((0,0))
```

récupère la valeur du pixel situé à la ligne 0 et à la colonne 0, autrement dit, dans le coin supérieur gauche.

Dans cet exemple, la variable `pixel` vaudra (158, 104, 58).



Attention aux parenthèses !

La fonction `getpixel()` possède un seul paramètre (qui est un tuple) et non deux paramètres (un paramètre par coordonnées). Ainsi, pour récupérer le pixel en (0,0) dans `Lena`, il faut bien écrire `lena.getpixel((0,0))` et non `lena.getpixel(0,0)`

Astuce

Comme pour les dimensions, on peut récupérer les composantes du pixel dans des variables séparées en une seule affectation. Par exemple, si on écrit :

```
r, g, b = lena.getpixel((0,0))
```

Alors `r` vaudra 158, `g` vaudra 104, et `b` vaudra 58.

Votre premier effet : miroir

L'effet « miroir » consiste à retourner l'image horizontalement, comme si on la regardait dans un miroir.


Implémentez une fonction :

```
def miroir(image):
```

qui crée et renvoie l'image miroir de l'image donnée en argument.

Attention : l'image miroir doit être une **nouvelle** image. Si vous travaillez directement sur l'image originale, vous allez écraser son contenu au fur et à mesure du traitement.

Exemple d'utilisation (sur `Lena`) :

Code	Résultat
<pre>lena = Image.open("lena.png") lena_miroir = miroir(lena) lena_miroir.show()</pre>	

N'hésitez pas à tester votre effet sur d'autres images !

Sauvegarde du résultat

Si vous souhaitez sauvegarder le résultat d'un traitement, vous pouvez utiliser la fonction décrite ci-après.

Etant donnée une image `img`, l'appel :

```
img.save(nom_fichier)
```

enregistre le contenu de l'image `img` dans le fichier dont le nom est spécifié en paramètre. Ce fichier est créé s'il n'existe pas, et son contenu est remplacé s'il existe.



Attention à ne pas écraser le fichier original !

Exemple : Pour enregistrer l'image miroir de Lena, vous pouvez écrire :

```
lena_miroir.save("lena_miroir.png")
```