Chapitre 5. Dessin d'images

Une image, en informatique, est un simple tableau à deux dimensions de points colorés (appelés pixels, *picture elements*).

Les coordonnées (x, y) d'un pixel expriment sa position au sein de l'image : x est son abscisse, en partant de la gauche de l'image, et y est son ordonnée, en partant du haut de l'image (à l'inverse de l'ordonnée mathématique, donc). Elles partent toutes deux de 0. Le schéma ci-dessous montre un exemple d'image en gris sur fond blanc, de 7 pixels de largeur et 4 pixels de hauteur, les abscisses vont donc de 0 à 6 et les ordonnées vont de 0 à 3.

0,0	1,0	2,0	3,0	4,0	5,0	6,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3

La couleur RGB (r, g, b) d'un pixel est la quantité de rouge (r pour *red*), vert (g pour *green*) et de bleu (b pour *blue*) composant la couleur affichée à l'écran. Le mélange se fait comme trois lampes colorées rouge, vert et bleue, et les trois valeurs r, g, b expriment les intensités lumineuses de chacune de ces trois lampes, exprimées entre 0 et 255. Par exemple, $(0, 0, 0)$ correspond à trois lampes éteintes, et produit donc du noir. $(255, 255, 255)$ correspond à trois lampes allumées au maximum [1], et produit donc du blanc. $(255, 0, 0)$ correspond à seulement une lampe rouge allumée au maximum, et produit donc du rouge. $(255, 255, 0)$ correspond à une lampe rouge et une lampe verte allumées au maximum, et produit donc du jaune, et ainsi de suite. Cela correspond très précisément à ce qui se passe sur vos écrans ! Si vous prenez une loupe, vous verrez que l'écran est composé de petits points (appelés *sous-pixels*) verts, rouges et bleus, allumés plus ou moins fortement pour former les couleurs.

Dans ce chapitre et le suivant, nous étudierons deux grands types d'opérations disponibles typiquement dans les logiciels de manipulation d'image : nous produirons d'abord des images de zéro (synthèse d'images), puis nous transformerons des images existantes (traitement d'images, ou filtres).

1. Le maximum de ce qu'elles peuvent produire. Même s'ils peuvent ainsi afficher des millions de couleurs différentes, nos écrans restent limités : https://www.youtube.com/watch?v=_NzVmtbPOrM

5.1 Mise en jambes

python permet bien sûr de manipuler des images, à l'aide de la *python Imaging Library* (PIL), qui est préinstallée sur vos machines. Nous vous fournissons un module `bibimages.py` qui permet d'utiliser plus simplement la bibliothèque PIL. Ce module est disponible en téléchargement sur le site https://moodle.u-bordeaux.fr/course/view.php?id=14677. Allez sur ce site, retrouvez dans la section du chapitre 5 le fichier `bibimages.py`, utiliser un clic droit dessus et utilisez "enregistrer sous" pour l'enregistrer à côté de vos autres fichiers python. Ce module comporte aussi une poignée de fonctions qui permettent de manipuler les images. Pour utiliser un module il faut commencer par l'*importer*, et toute session de travail sur les images doit commencer par la phrase magique :

```
from bibimages import *
```

Voici un résumé des fonctions que nous utiliserons dans ce chapitre :

`ouvrirImage(nom:str) -> image`	Ouvre le fichier *nom* et retourne l'image qu'il contient, par exemple : `img = ouvrirImage("teapot.png")`
`nouvelleImage(large:int, haut:int) -> image`	Retourne une image de taille $large \times haut$, initialement noire, par exemple : `img = nouvelleImage(300, 200)`
`afficherImage(img:image)`	Affiche l'image *img*, par exemple : `afficherImage(img)`
`colorierPixel(img:image, x:int, y:int, (r,g,b))`	Peint le pixel (x, y) dans l'image *img* de la couleur (r, g, b), par exemple : `colorierPixel(img, 10, 10, (255, 0, 0))`
`largeurImage(img:image)`	Retourne le nombre de colonnes de pixels contenues dans *img*, par exemple : `l = largeurImage(img)`
`hauteurImage(img:image)`	Retourne le nombre de lignes de pixels contenues dans *img*, par exemple : `h = hauteurImage(img)`

Exercice 5.1.1 TP Après avoir suivi le début de cette page pour mettre en uvre `bibimages`, exécuter les instructions suivantes :

```
monImage = nouvelleImage(300,200)
colorierPixel(monImage, 10, 10, (255,255,255))
afficherImage(monImage)
```

Expliquer ce qu'effectue chaque instruction, et observer l'image produite. Confirmer ainsi le sens dans lequel fonctionnent les coordonnées.

Peindre le pixel de coordonnées $(250, 150)$ en vert (n'oubliez pas d'appeler de nouveau `afficherImage(monImage)` pour afficher le résultat). Peindre le pixel de coordonnées $(50, 150)$ en violet.

Que contiennent les variables `l` et de `h` après avoir exécuté les instructions suivantes ?

```
l = largeurImage(monImage)
h = hauteurImage(monImage)
```

Récupérez l'image de la célèbre théière de l'Utah (voir Wikipedia) sur la page des supports du site

https://moodle.u-bordeaux.fr/course/view.php?id=14677
et sauvegardez-la à côté de vos fichiers `.py`. Exécuter les instructions suivantes :

```
theiere = ouvrirImage("teapot.png")
afficherImage(theiere)
```

Télécharger au moins une autre image à partir de la même page, et vérifiez que vous pouvez également l'ouvrir.

Vous pourrez bien sûr, pour tous les exercices, utiliser d'autres images venant d'Internet ou d'ailleurs (pour autant que les licences sous lesquelles lesdites images sont placées vous le permettent !).

Rappel :

Une chaîne de caractères est une suite de valeurs numériques élémentaires, dont chacune représente un caractère. En python, on définit une chaîne de caractères comme un ensemble de caractères placés entre deux caractères "guillemets" ou "apostrophe" : `"Bonjour"`, `'tout le monde'`. De nombreuses fonctions permettent de manipuler des chaînes de caractères : la fonction `print` permet d'afficher une chaîne de caractères, l'opérateur "`+`" permet de concaténer deux chaînes pour en former une nouvelle, etc.

Ici, `"teapot.png"` est une chaîne de caractères contenant le nom du fichier à ouvrir. python ne cherche pas à comprendre ce qu'il y a entre les guillemets. Si par contre on oublie les guillemets, python va chercher une variable appelée `teapot`, ce qui n'est pas du tout ce que l'on veut.

5.2 Tracés de segments

Exercice 5.2.1 Quelles sont les coordonnées des pixels d'une ligne horizontale au milieu d'une image de hauteur `hauteur` et de largeur `largeur` ? Bien spécifier le premier et le dernier pixel. Laquelle parmi les deux coordonnées reste constante et laquelle est variable ?

Écrire une fonction `ligneHorizontaleBlancheAuMilieu(img)` qui utilise une boucle `for` et `range` pour dessiner une ligne horizontale blanche au milieu de l'image `img` (séparant donc l'image donnée en deux parties). Testez-la sur la théière, sur une image noire (générée par un appel à `nouvelleImage`), ainsi que sur au moins une autre image. Notez que notre fonctionne ne *retourne rien* : en effet, elle *modifie* l'image passée en paramètre, elle n'a donc pas besoin de la retourner.

Exercice 5.2.2 Écrire une fonction `ligneHorizontaleAuMilieu(img,c)` qui a donc un paramètre en plus, `c`, qui permet ainsi de désormais choisir la couleur de la ligne lors de l'appel. En vous inspirant de la fonction précédente, il suffit de remplacer, dans le corps de la fonction, la couleur blanche `(255,255,255)` par la couleur `c`. Tester :

```
ligneHorizontaleAuMilieu(monImage, (255,0,0))
afficherImage(monImage)
ligneHorizontaleAuMilieu(monImage, (0,255,0))
afficherImage(monImage)
```

La fonction récupère donc l'ensemble des trois valeurs r, g, et b dans le seul paramètre `c`, qu'elle peut passer tel quel à `colorierPixel`.

Exercice 5.2.3 Écrire une fonction `ligneHorizontale(img,c,y)` qui dessine, dans une image `img` donnée, une ligne horizontale de couleur `c` à la distance `y` du haut de l'image.

Vous pouvez l'appeler plusieurs fois sur une même image avec des paramètres différents pour constater le résultat. C'est tout l'intérêt d'avoir ajouté des paramètres : on n'a pas besoin de modifier la fonction pour obtenir un dessin évolué.

Que se produit-il si l'on appelle la fonction `ligneHorizontale(img,c,y)` avec une valeur de `y` qui dépasse la hauteur de l'image `img` ?

Améliorer le code de votre fonction afin de tester d'abord si la valeur de `y` est valide. Si ce n'est pas le cas, votre fonction devra ne rien dessiner.

Donner une nouvelle version de la fonction `ligneHorizontaleAuMilieu2(img,c)` qui dessine une ligne horizontale de couleur `c` au milieu de l'image `img` en ne faisant qu'appeler la fonction `ligneHorizontale(img,c,y)`.

Exercice 5.2.4 En faisant appel à la fonction `ligneHorizontale`, écrire une fonction `grilleHorizontale(img, c, d)` qui dessine une grille de lignes horizontales espacées par `d` pixels.

Note : profitez de la fonction `range(debut,fin,pas)`, qui est toute prête à vous fournir la liste exacte des ordonnées concernées.

Par exemple, `grilleHorizontale(monImage,(255,255,0),10)` dessinerait dans `monImage` des lignes jaunes avec pour ordonnées 0, 10, 20, 30, etc.

Exercice 5.2.5 Écrire une fonction `rectangleCreux(img,x1,x2,y1,y2,c)` qui dessine les côtés d'un rectangle de couleur `c`, les coins du rectangles étant les pixels de coordonnées $(x1, y1)$, $(x2, y1)$, $(x1, y2)$, $(x2, y2)$. Il s'agit bien sûr de dessiner quatre lignes formant les côtés du rectangle. Testez-la plusieurs fois avec des coordonnées différentes et des couleurs différentes, sur une image noire et sur la photo de théière.

Que se passe-t-il si l'un des paramètres dépasse de la taille de l'image ? Est-il facile de corriger le problème ?

Exercice 5.2.6 Qu'effectue la fonction suivante ? N'hésitez pas à la copier depuis le PDF disponible sur le site du cours et à la coller dans Python Tutor.

```
def remplir(img, c):
    l = largeurImage(img)
    h = hauteurImage(img)
    for x in range(l):
        for y in range(h):
            colorierPixel(img, x, y, c)
```

Décrire précisément ce qu'effectue chaque ligne, testez-la. Est-ce que les pixels sont coloriés ligne par ligne ou colonne par colonne ? Si le premier pixel colorié est celui de coordonnées $(0, 0)$, quelles sont les coordonnées du pixel colorié en deuxième, $(0, 1)$ ou $(1, 0)$?

Mêmes questions pour la version suivante :

```
def remplirBis(img, c):
    l = largeurImage(img)
    h = hauteurImage(img)
    for y in range(h):
        for x in range(l):
            colorierPixel(img, x, y, c)
```

Définir une fonction `remplir2(img,c)` produisant le même résultat, en utilisant la fonction `ligneHorizontale`.

Serait-il possible possible d'obtenir le même effet avec la fonction `grilleHorizontale` ?

Exercice 5.2.7 En vous inspirant très largement de la fonction `remplir`, écrivez une fonction `rectanglePlein(img,x1,x2,y1,y2,c)` qui dessine un rectangle plein de couleur `c` dont les coins sont les pixels de coordonnées $(x1, y1)$, $(x2, y1)$, $(x1, y2)$, $(x2, y2)$.

Exercice 5.2.8 On veut colorier toute une image `img` en gris. On a commencé par taper ceci :

```
#code a completer
c = (127,127,127)
l = largeurImage(img)
h = hauteurImage(img)
```

et il s'agit maintenant de parcourir toute l'image pour la colorier. Parmi les codes suivants, lesquels sont corrects ? Rayer les codes erronés.

```
for x in range(h):
  for y in range(l):
    colorierPixel(img, x, y, c)
```

```
for x in range(1,l,1):
  for y in range(1,h,1):
    colorierPixel(img, x, y, c)
```

```
for y in range(h):
  for x in range(l):
    colorierPixel(img,x, y, c)
```

```
for x in range(1,l+1):
  for y in range(1,h+1):
    colorierPixel(img, x, y, c)
```

```
for x in range(l) and y in range(h):
  colorierPixel(img, x, y, c)
```

Exercice 5.2.9 Écrire une fonction `remplirDegradeGris(img)` qui remplit l'image `img` avec un dégradé de gris depuis du noir en haut à du blanc en bas : tous les pixels de la ligne tout en haut de l'image doivent être noirs, et tous les pixels de la ligne tout en bas de l'image doivent être blancs, et les pixels des lignes intermédiaires, d'un niveau de gris intermédiaire, en dégradé.

Que suffit-il de changer pour que le dégradé soit de la gauche vers la droite ?

Que suffit-il de changer pour que le dégradé soit du bas vers le haut ?

5.3 Exercices de révisions et compléments

Exercice 5.3.1 Écrire une fonction `ligneVerticale(img,c,x)` qui dessine, dans une image `img` donnée, une ligne verticale de couleur `c` à la distance `x` du bord gauche de l'image.

En faisant appel à celle-ci, écrire une fonction `grilleVerticale(img,c,d)` qui dessine une grille de lignes verticales espacées par `d` pixels.

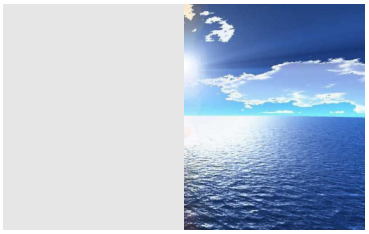
Exercice 5.3.2 Écrire une fonction `remplirRougeVertJaune(img)` qui remplit l'image `img` avec un dégradé de couleurs : le coin en haut à gauche doit être noir, le coin en bas à gauche doit être vert, le coin en haut à droite doit être rouge, et le coin en bas à droite doit être jaune (le jaune est le mélange à part égale de rouge et de vert).

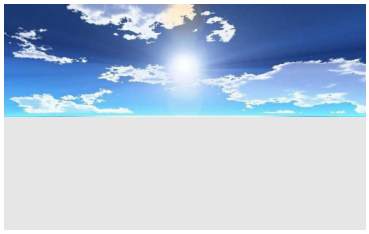
De la même façon, écrivez `remplirRougeBleuViolet(img)` et `remplirVertBleuCyan(img)`

Exercice 5.3.3 En faisant appel aux fonctions `grilleHorizontale(img,c,d)` et `grilleVerticale(img,c,d)`, écrire une fonction `grilleCarree(img,c,d)` qui dessine dans une image une grille carrée d'espacement `d` pixels. Tester avec plusieurs images et plusieurs valeurs de `d`.

Exercice 5.3.4 (extrait DS 2015-16) On souhaite masquer la moitié d'une image à l'aide d'une couleur. Voici un exemple :


image initiale


partie gauche masquée


partie inférieure masquée

1. Écrire la fonction `masquerGauche(img,c)` qui remplit de la couleur `c` la moitié gauche de l'image `img` et laisse inchangé le reste de l'image.

2. Écrire la fonction `masquerBas(img,c)` qui remplit de la couleur `c` la moitié inférieure de l'image `img` et laisse inchangé le reste de l'image.

Exercice 5.3.5 Écrire une fonction `diagonale(img)` qui dessine en blanc la ligne diagonale entre le coin en haut à gauche et le coin en bas à droite de l'image. Tester avec différentes tailles d'image, et corriger l'erreur que vous avez commise.

Exercice 5.3.6 On souhaite ajouter des lignes verticales noires à une image.

Écrire une fonction `emprisonner(img, nbBarreaux, epaisseur)` qui ajoute `nbBarreaux` lignes verticales d'`epaisseur` pixels et réparties de façon homogène dans l'image.

Exercice 5.3.7 En vous souvenant du schéma du cercle trigonométrique pour écrire l'équation paramétrique de la courbe du cercle, écrire une fonction `cercle(img,x,y,r)` qui dessine un cercle de rayon r dont le centre a pour coordonnées (x, y).

Note : Les outils trigonométriques (`cos`, `sin`, `pi`, ...) sont disponibles en tapant :

```
from math import *
```

Exercice 5.3.8 On souhaite masquer une image en diagonale à 45 degré à l'aide d'une couleur. Voici un exemple :


image initiale


partie diagonale masquée

1. Écrire la fonction `masquerDiagonale(img, c)` qui remplit de la couleur `c` le triangle inférieur gauche de l'image `img` et laisse inchangé le reste de l'image.

2. Est-ce que votre fonction est correcte pour une image qui serait plus haute que large ? Sinon, corrigez.

Exercice 5.3.9 Écrire une fonction `remplirDegradeDiagonaleBleu(img)` qui remplit l'image `img` avec un dégradé de bleu en diagonale : le coin en haut à gauche doit être noir, le coin en bas à droite doit être bleu.

Écrire une fonction `remplirDegradeDiagonaleBleuInverse(img)` qui remplit l'image `img` avec un dégradé de bleu en diagonale dans l'autre sens : le coin en haut à gauche doit être bleu, le coin en bas à droite doit être noir.

En combinant les formules utilisées pour `remplirRougeVertJaune` et pour `remplirDegradeDiagonaleBleu`, écrire une fonction `remplirCouleurs(img)` qui remplit l'image `img` avec un dégradé entre les différentes couleurs : le coin en haut à gauche doit être bleu, le coin en bas à gauche doit être vert, le coin en haut à droite doit être rouge, et le coin en bas à droite doit être jaune.

5.4 L'essentiel du chapitre

On peut ou bien ouvrir une image existante dans un fichier :
`monImage = ouvrirImage("fichier.png")`
ou bien créer une image :
`monImage = nouvelleImage(300, 200)`

Dans les deux cas on peut ensuite colorier des pixels :
`colorierPixel(monImage, 50, 50, (100, 0, 0))`

Pour connaitre la taille de l'image, on utilise les fonctions `largeurImage` et `hauteurImage`.

Ainsi, une fonction typique qui trace un dessin dans une image est :

```
def f(img, n):
    l = largeurImage(img)
    h = hauterImage(img)
    for x in range(l):
        y = [...]
        c = [...]
        colorierPixel(img, x, y, c)
```

Et on l'utilise par exemple ainsi :

```
monImage = nouvelleImage(300, 200)
f(monImage)
afficherImage(monImage)
```

Note : la fonction `f` prend une image, et non le nom d'un fichier. Cela permet ainsi d'appeler `f` plusieurs fois, possiblement avec la même image et différents autres arguments, les effets des appels s'accumulant dans l'image, ou possiblement avec d'autres images, etc. à volonté.