ESTIMATION DE LA LUMINOSITÉ D'UNE ÉTOILE VARIABLE
à partir d'une photographie.

Exercices préparatoires

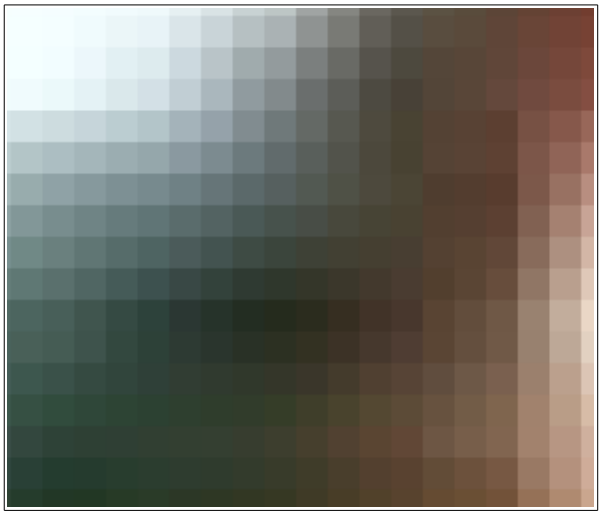
1. Lecture de la documentation de la bibliothèque "scipy"

En plus des bibliothèques de fonctions déjà nommées (matplotlib pour les graphismes, PIL pour les images et numpy pour les tableaux de nombres), il est nécessaire d'ajouter une bibliothèque de fonctions mathématiques pour le calcul "scientifique".
Pour le langage Python, elle résulte du projet "scipy" (SCIences PYthon).
Voir https://fr.wikipedia.org/wiki/SciPy.
On y trouve des modules de fonctions pour le traitement du signal et le traitement d'images.
Traitement d'image ! c'est ce qui m'intéresse...
La documentation, en anglais hélas, pour l'ensemble de ces bibliothèques est accessible à partir de : http://www.scipy.org/docs.html
Celle qui contient les fonctions citées dans cette page http://docs.scipy.org/doc/scipy/reference/
Si vous avez la curiosité de cliquer sur ce lien, vous trouverez ceci : Multi-dimensional image processing (`scipy.ndimage`).
Le texte suivant rend compte de la lecture et de l'expérimentation ce certaines fonctions présentées à cette page http://docs.scipy.org/doc/scipy/reference/ndimage.html

L'anglais utilisé dans ces documentations techniques est loin de celui que nous avons pu apprendre : le vocabulaire est assez restreint, avec des mots qui reviennent sans cesse (ce sont les mots techniques de la discipline).
Le plus souvent, les verbes sont au présent et les structures grammaticales sont simplistes.

2. Les images sont des tableaux de données à deux dimensions.

Une image JPEG en gamme de gris, contient un seul tableau de nombres en deux dimensions (comparable à une grille de mots croisés ou à un damier).
Dans chaque case (cellule, position définie par les coordonnées y et x), on trouve une valeur allant de 0 à 254.



Une image JPEG en couleur contient trois tableaux comparables au précédent : un pour le rouge, un pour le vert, un pour le bleu.

Une image FITS peut être comparée à ce qui précède, sauf que les valeurs contenues dans les cellules peuvent être définies de façon plus large et plus souple.
L'image ci-contre montre un extrait d'image JPEG, fortement zoomée, où chacune des cellules qui la composent apparaît individuellement.

Pour résumer : travailler sur l'image numérique, c'est travailler sur des tableaux de nombres à deux dimensions.

3. Créer des tableaux à deux dimensions avec Python et numpy

Un tableau doit contenir quelque chose.
Commençons par créer un tableau carré de 10 sur 10 :

```
import numpy as np
a = np.zeros((10,10), dtype=np.int8)
print a
```

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

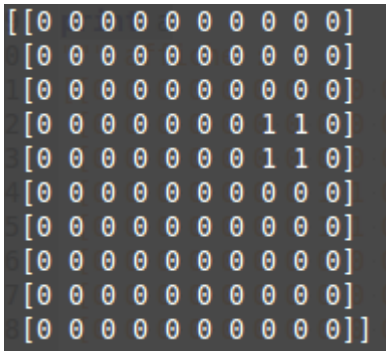
Il imprime le tableau affiché à droite :
Commentaires sur le code Python :

`import numpy as np`	`numpy` est la bibliothèque qui "sait" créer les tableaux en deux dimensions. Par la suite l'usage de la bibliothèque sera référencée par le "petit nom" `np`
`a = np.zeros((10,10), dtype=np.int8)`	Création du tableau `a` est créé. Il en rempli de zéros (`np.zeros`). Chacune des cellules peut contenir une valeur allant de 0 à 254 (`dtype=np.int8`)
`print a`	Liste le contenu du tableau créé.

4. Introduire une série de valeurs dans le tableau.

Nous voulons mettre des "1" dans les colonnes 7 et 8 des lignes 3 et 4.
En Python cela s'écrit :

```
a[3:5,7:9] = 1
print a
```



Qui imprime le tableau affiché à droite :

Commentaires sur le code Python :

Les lignes et colonnes sont numérotées à partir de 0.

3 : 5 veut dire que l'on part de la ligne 3 et que l'on s'arrête avant la ligne 5

7 : 9 veut dire que l'on commence à la colonne 7 et que l'on s'arrête avant la colonne 9

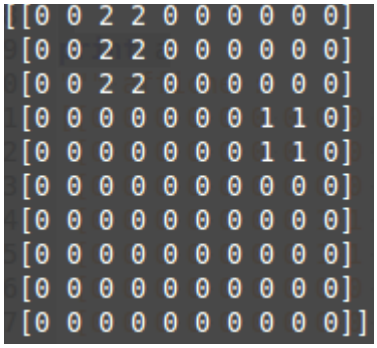
Les crochets [] et la virgule font partie de la syntaxe Python.

5. Introduire une seconde série de valeurs dans le tableau.

Nous voulons mettre des "2" dans les colonnes 2 et 3 des lignes 30 et 1.

Sans surprise, en Python cela s'écrit :

```
a[0:3,2:4] =2
print a
```



Qui imprime le tableau affiché à droite :

En résumé :

- Nous considérons comme paquet de données toutes les cellules contigues qui contiennent autre chose que zéro.
- Nous venons de créer deux paquets de données (le paquet qui contient des 1 et le paquet qui contient des 2).
- On peut imaginer que chacun de ces paquets est une étoile dans une image. Dans ce qui suit, paquet de données = étoile et tableau de données = image.
- Les valeurs contenues dans l'image fictive (0, 1 ou 2) sont les valeurs des "pixels" de l'image.

Par la suite, nous voulons faire ceci :

- Localiser le centre de chaque *étoile* dans notre *image*.
- Totaliser les valeurs des *pixels* de chacune des *étoiles*, de *l'image* ce qui permettra, par la suite, de comparer leurs magnitudes.

6. Utilisation de scipy.ndimage

Cette bibliothèque a été citée plus haut.
Elle contient des fonctions essentielles appelées "center_of_mass", "label", "sum".
Brièvement, on peut les décrire ainsi :

Fonctions	Utilité
center_of_mass	Localise en y et x le centre d'un groupe de pixels (centre d'une étoile)
sum	Additionne tous les pixels d'une "étoile" (paquet de données)
label	Fonction nécessaire pour faire fonctionner les deux précédentes

7. Trouver le centre des étoiles.

Pour accéder aux fonctions décrites, il faut d'abord les importer dans Python :

```
from scipy.ndimage import  center_of_mass, label, sum
```

Le code tient en deux lignes, même si son commentaire sera plus long.

```
lbj,nb = label(a)
print "Centre des masses de points (y et x) : ",center_of_mass(a,lbj,[1,2])
```

Commentaires sur le code :

lbj,nb = label(a)	Le tableau de données s'appelle a Il est passé en paramètre à la fonction label() Celle ci retourne deux valeurs, dont l'une lbj nous est nécessaire pour la suite. nb ne sera pas utilisé.
center_of_mass(a,lbj,[1,2])	center_of_mass est la fonction qui fait le travail. En plus des paramètres a et lbj déjà cités, on doit lui passer les numéros des paquets à "traiter" [1,2]
print	Affiche les valeurs retournées par la fonction.

Résultat affiché par le programme :
[(1.0, 2.5), (3.5, 7.5)]
Ce qui s'interprète ainsi :
Coordonnées du centre de la première *étoile* : y =1,0 et x = 2,5
Coordonnées du centre de la seconde *étoile* : y =3,5 et x = 7,5

8. Additionner les pixels des étoiles.

sum(a,lbj,[1,2])
Qui donne pour résultat :
[12. 4.]
Ce qui s'interprète ainsi :
Somme des *pixels* de la première *étoile* : 12
Somme des *pixels* de la seconde *étoile* : 4

9. Conclusion

A partir de la maquette présentée ici, nous sommes parvenus à obtenir deux types d'informations que nous recherchions dans les notes précédentes :
- La position de chacune des étoiles (ici par l'intermédiaire de son centre).
- La somme des valeurs des pixels pour chacune, ce qui permettra de comparer les magnitudes.

Ces résultats ont été obtenus sans faire appel à des logiciels annexes et coûteux, mais uniquement par du code écrit en Python.
Ce code est extrêmement court et nous n'avons pas eu à coder des fonctions mathématiques compliquées, sources d'erreurs multiples pour un codeur débutant.
La contrepartie, non négligeable, est qu'il est nécessaire de lire une documentation abondante, en anglais, mais facilement et gratuitement accessible.

Il faut maintenant voir comment adapter ce code au traitement d'images réelles.