

DS n° 04 – Rackham le Rouge

- Faire tous les exercices dans un fichier NomPrenom.py à sauvegarder,
- mettre en commentaire l'exercice et la question traités (ex : # Exercice 1),
- ne pas oublier pas de commenter ce qui est fait dans votre code (ex : # Je crée une fonction pour calculer la racine d'un nombre),
- compte tenu du fait que l'épreuve est à distance, il n'est pas possible de demander un déblocage pour une question,
- il faut vérifier avant de déposer le fichier (sur <https://www.ptsi-dorian.net/depot-de-fichiers>), que le code peut s'exécuter et qu'il affiche les résultats que vous attendez.

Les documents ressources pour ce sujet sont disponibles à l'adresse suivante :
<https://github.com/Costadoat/Informatique/raw/master/DS/2023-2024/DS04/ressources.zip>
 Vous devez extraire les fichiers du .zip ou les télécharger 1 par 1 depuis le dossier github.

1 La maquette de la Licorne

Dans une de ses aventures, Tintin doit rechercher « Le trésor de Rackham le Rouge ». Nous allons, avec cette épreuve, l'aider dans cette recherche.

Dans le livre précédent « Le secret de la Licorne », Tintin achète une maquette de bateau sur un marché.

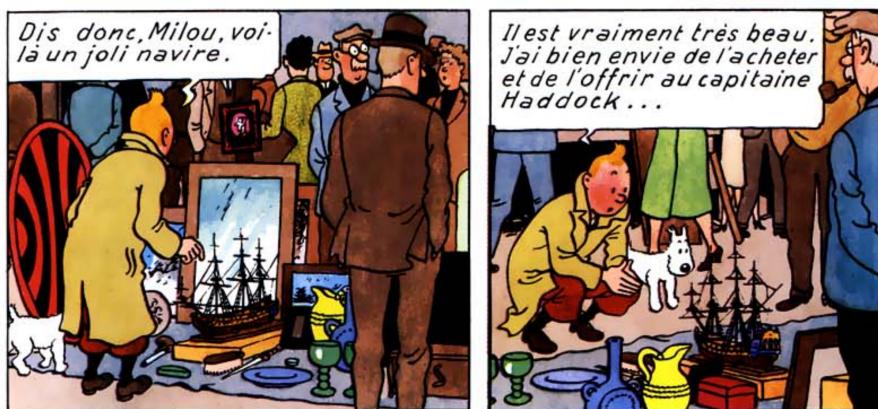


FIGURE 1 – Tintin achète une maquette de la Licorne au marché

Il s'agit d'une maquette de la Licorne, un bateau ayant appartenu au chevalier de Hadoque, un ancêtre du Capitaine Haddock, et il en existe 3 parfaitement identiques.

Après quelques péripéties, il découvre qu'un parchemin se cachait à l'intérieur du mat creux des trois maquettes.

Ces parchemins correspondent aux fichiers `parchemin1.png`, `parchemin2.png` et `parchemin3.png`.

Les images sont codées en RGBA, avec **des valeurs allant de 0 à 1**.



FIGURE 2 – Découverte du parchemin

Question 1 A l'aide de la bibliothèque `matplotlib.image` et `matplotlib.pyplot`, **ouvrir** `parchemin1.png`, `parchemin2.png` et `parchemin3.png` et les stocker dans 3 variables (`parchemin1`, `parchemin2` et `parchemin3`). **Afficher** seulement `parchemin1.png` à l'aide d'une commande `imshow`.

Les parchemins font apparaître un message codé quand on les superpose.

Pour cela, nous allons additionner `parchemin1`, `parchemin2` et `parchemin3` et afficher l'image qui en résulte.

Il y a pourtant un problème avec cette addition. Pour certains pixels, la valeur dépasse 1 (on rappelle que 1 est la valeur maximale).

Question 2 **Créer** la variable `parchemin` obtenue en faisant la somme des 3 variables définies à la question 1. Parcourir chaque pixel de l'image afin de **saturer** les valeurs à 1. Ex : si la valeur RGBA du pixel i,j a pour valeur [1.2,0.6,0,1], on la remplace par [1,0.6,0,1].

Il existe plusieurs méthodes pour effectuer cette opération, elles sont toutes acceptées.

Le message secret apparaît et il indique les coordonnées de l'épave de la licorne qui est sensée contenir le trésor.

2 Recherche de l'épave du bateau

La suite va tenter de déterminer où se trouve l'épave à partir de ces coordonnées. Si vous n'avez pas réussi la première partie, vous pouvez essayer de reconstituer les coordonnées à partir des 3 parchemins.

Les coordonnées présentes sur le message sont de la forme :

AA°BB'CC"W DD°EE'FF"N

Ces coordonnées DMS, se lisent comme suit : AA degrés, BB minutes (60ème de degré), CC secondes (60ème de minute) vers l'ouest (W) à partir du méridien origine. DD degrés, EE minutes (60ème de degré), FF secondes (60ème de minute) vers le nord (N) à partir de l'équateur. Où AA, BB, CC, DD, EE, FF sont des entiers.

Il est alors possible de convertir ces coordonnées en coordonnées DD qui ne contiennent que les valeurs en degré mais sous forme de float signés. Cela reviendrait, pour l'expression du temps, à convertir 12h30min en 12.5h. Le signe vient du sens de déplacement, « + » si le déplacement est vers le Nord ou vers l'Est et « - » si le déplacement est vers le Sud ou vers l'Ouest.

Question 3 **Proposer** une fonction `dmstodd(coord,dir)` qui retourne un float au format DD à partir des coordonnées DMS. **Montrer** que `dmstodd([20,37,42], 'N')` retourne 20.6283333333334.

Le trésor de Rackham le Rouge

Après de longues fouilles, les explorateurs se rendent compte que l'épave ne se trouve pas à l'endroit prévu. Alors qu'ils s'apprêtent à rentrer bredouilles, Tintin a une idée.



FIGURE 3 – Tintin comprend qu'ils se sont trompés dans le calcul des coordonnées

En effet, du temps du chevalier de Haddock, le méridien d'origine n'était pas celui de Greenwich (près de Londres) mais celui de Paris, situé $2^{\circ}20'13''$ à l'est de celui de Greenwich.

Question 4 **Proposer** les nouvelles coordonnées de l'épave au format DD et **déterminer** quelle distance doit parcourir le bateau afin de regagner l'emplacement réel de l'épave.

On donne la fonction `distance` qui retourne la distance en km entre deux points A et B désignés par leurs coordonnées (Latitude, Longitude) exprimées au format DD :

```
def distance(pointA,pointB):
    rayon_terre=6361
    phi1=pointA[0]*np.pi/180
    phi2=pointB[0]*np.pi/180
    deltalambda=(pointB[1]-pointA[1])*np.pi/180
    return np.arccos(np.sin(phi1)*np.sin(phi2) \
        +np.cos(phi1)*np.cos(phi2)*np.cos(deltalambda))*rayon_terre
```

L'épave était bien là, mais malgré des explorations poussées, ils ne trouvent aucun trésor. Ils trouvent par contre des documents signifiant que le chevalier de Hadoque était propriétaire du Château de Moulinsard. Or celui-ci est à vendre et Tintin et le capitaine Haddock décident de visiter ses caves.

Passons à la suite pour découvrir ce que contenait ce trésor.



FIGURE 4 – Tintin et la capitaine trouvent le trésor au château de Moulinsart

3 Le trésor de Rackham le Rouge

Le trésor est magnifique, il contient des rubis (rouges), des saphirs (bleus) et des émeraudes (vertes). L'image tresor.png présente dans le dossier de travail contient une photo en négatif (couleurs inversées) d'une partie de ce trésor.

L'image en couleurs réelles est donnée dans le dossier de travail pour ne pas vous bloquer pour la suite.



FIGURE 5 – Négatif photo du trésor

Question 5 A l'aide d'un script python, inverser les couleurs de l'image tresor.png et afficher l'image avec les bonnes couleurs. Compter à la main, le nombre de rubis, de saphirs et d'émeraudes. Le décompte les bijoux sans image inversée ne rapporte pas de points.

Le trésor étant très important, on souhaite automatiser le comptage des bijoux.
Pour cela, on souhaite suivre la procédure suivante :

- 1 Prendre en photo une partie des bijoux,
- 2 Extraire, par identification des couleurs, les bijoux d'une certaine couleur,
- 3 Compter le nombre de bijoux de chaque couleur présents sur la photo.

3.1 Prendre la photo

Cette partie est réalisée par un photographe.

3.2 Extraire les bijoux par couleur

La solution pour isoler les couleurs revient à conserver les pixels dont la couleur est « proche » du rouge, puis du vert et enfin du bleu. Cela donnerait pour la figure 5 inversée les 3 images de la figure 6.

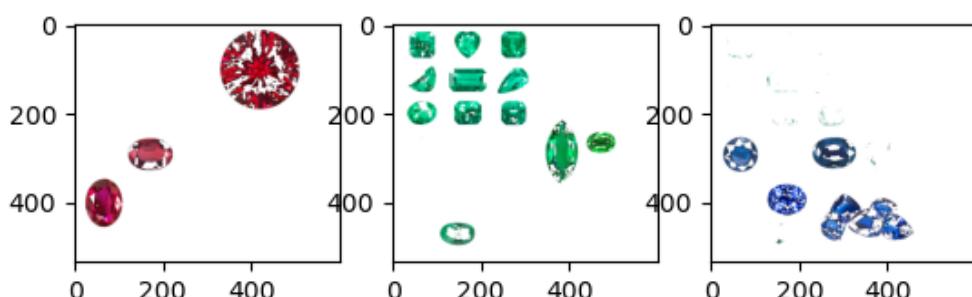


FIGURE 6 – Images générées à partir de la photo de la figure 5

Pour mesurer cette proximité, nous allons considérer un pixel comme un vecteur de coordonnées (R, G, B, A) (R : teneur en rouge [0..1], G : teneur en vert [0..1], B : teneur en bleu [0..1], A : opacité [0..1]).

Deux pixels A et B seront considérés proches si la norme de leur différence est inférieure à un seuil s .

$$\sqrt{(R_A - R_B)^2 + (G_A - G_B)^2 + (B_A - B_B)^2 + (A_A - A_B)^2} < s$$

Question 6 Écrire une fonction `norme(vec)` qui prend en entrée un vecteur, sous la forme d'une liste de dimension n , et retourne sa norme quel que soit n . Montrer que `norme([1, 2, 3])` renvoie 3.7416573867739413.

Question 7 Recopier et compléter le code suivant afin que tous les pixels pour lesquels la norme est supérieure au seuil soit coloriés en blanc.

```

import copy

seuil=0.8
red,green,blue=[1,0,0,1],[0,1,0,1],[0,0,1,1]
img_coul=[]

for i in range(3):
    img_coul.append(image_couleurs.copy())
for idx,couleur in enumerate([red,green,blue]):
    for i in range(taille[0]):
        for j in range(taille[1]):
            if norme(img_coul[idx][i,j]-couleur)>seuil:
                .....
f, axarr = plt.subplots(1,3)
for i in range(3):
    axarr[i].imshow(img_coul[i])
plt.show()

```

3.3 Compter les bijoux

Afin de compter le nombre de bijoux, nous allons dans cette partie **détecter** les contours des bijoux puis **parcourir** l'image à l'aide d'un parcours de graphe. Cette dernière étape qui n'est pas demandée permettra de compter le nombre de cycles dans la figure et d'en déduire le nombre de bijoux.

Afin de détecter les contours sur la figure, nous allons dans un premier temps convertir la figure en noir et blanc.

Question 8 Coder la fonction `convert_nb(image)` qui convertit une image (en couleur) en niveaux de gris.

Question 9 Coder la fonction `contour(image_nb)` qui convertit une image (en niveaux de gris) en contours.

Question 10 Afficher les images obtenues à l'aide du code suivant.

```

f, axarr = plt.subplots(1,3)
axarr[0].imshow(image_couleurs)
axarr[0].title.set_text('Image originale')
axarr[1].imshow(image_nb)
axarr[1].title.set_text('NB')
axarr[2].imshow(contours)
axarr[2].title.set_text('Contour')
plt.show()

```

Question 11 Donner les limites de la méthode utilisée, quelle autre approche aurait pu être utilisée pour ce cas de figure ?

FIN