EXAMEN DE FIN DE TRIMESTRE 2

Matière : INFORMATIQUE

Classes : 2ème Année BG **Durée : 2 h**

DOCUMENTS NON AUTORISES

Exercice 1 :

Rappel :

- Une séquence ADN est formée des quatre bases A, G, C, T.
- Une région codante : commence par un codon **start** : ATG et se termine par un codon **stop** : TAA, TAG ou TGA

1. Importez la méthode **seq** du module **biopython** et définir manuellement une séquence **brin1** d'ADN totalisant 15 éléments composée d'un codon **start**, d'une suite d'éléments de l'ensemble des bases et un codon **stop** quelconque.
2. Donnez la commande python permettant de fournir la séquence **brin2** complémentaire de **brin1**.

Dans la suite de l'exercice nous allons écrire les fonctions permettant de générer, enregistrer et lire une séquence valide.

3. Définissez deux listes :
   - BASES_ADN : contenant les quatre bases
   - STOP : contenant les différents codons stop
4. Écrire une fonction **`gen_brins`** qui génère aléatoirement un brin d'ADN. On pourra choisir aléatoirement un codon STOP pour terminer le brin ou la partie codante. Cette fonction prendra comme argument le nombre total d'éléments de la séquence. On vérifiera que la longueur souhaitée est un multiple de trois. L'entête de la fonction est : **`def gen_brins(nbases)`**
5. Écrire une fonction **`write_file(brin, fichier)`** qui écrit le brin d'ADN dans un fichier.
6. Écrire une fonction qui lit un brin d'ADN à partir d'un fichier.
7. Écrire une fonction qui retourne la proportion de la chaîne <CH> dans la séquence <brin>.

Exercice 2

Traitement d'images :

Les images que nous allons traiter sont des images dites matricielles. Elles sont composées d'une matrice (tableau numpy) de points colorés appelés pixels. Chaque pixel est un triplet de nombres entre 0 et 255 : un nombre pour chaque couleur primaire rouge, vert, bleu.
On utilise ici la synthèse additive des couleurs : le triplet [0, 0, 0] correspond à un pixel noir alors qu'un pixel blanc est donné par [255, 255, 255]. Un pixel « pur rouge » est codé par [255, 0, 0].
On notera que le pixel de coordonnées (0, 0) est conventionnellement situé en haut et à gauche de l'image.

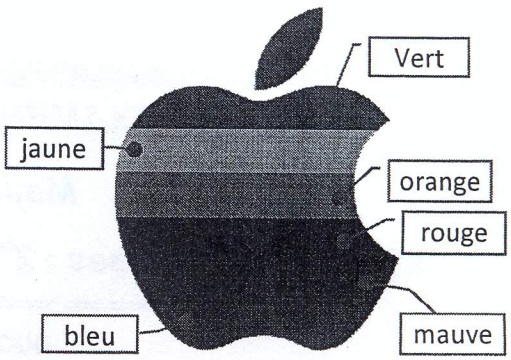

image A (en couleur)

NB : En python, nous traitons les images comme étant des matrices <ndarray> du module numpy
Il est **interdit** d'utiliser des fonctions prédéfinies **non présentes** dans l'annexe.

1. Une image négative est une image dont les couleurs ont été inversées par rapport à l'originale ; par exemple le rouge devient cyan, le vert devient magenta, le bleu devient jaune et inversement. Les régions sombres deviennent claires, le noir devient blanc. Pour cela il suffit d'inverser les niveaux de chacune des couleurs primaires : un pixel initialement à [120, 10, 250] deviendra [135, 245, 5]. C'est-à-dire que le ton d'un pixel de la nouvelle image est 255 – v, où v est le ton du pixel correspondant de l'image d'origine.

Écrire une fonction ***negatif(img)*** retournant l'image négative de l'image passée en paramètre.

2. Dans une image en niveaux de gris, les trois composantes R, V, B de chaque pixel ont la même valeur. Celle-ci vaut 0 pour un pixel noir, 255 pour un pixel blanc.

L'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités de couleurs fondamentales, mais une moyenne pondérée. La formule standard donnant le niveau de gris en fonction des trois composantes est :

gris = partie entière de (0.299 * rouge + 0.587 * vert + 0.114 * bleu)

Ecrire une fonction ***niveau_gris(im)*** renvoyant une nouvelle image, en niveau de gris. Les trois niveaux R, V, B d'un pixel sont égaux au niveau de gris donné par la formule ci-dessus. On aura besoin de la fonction round() pour calculer la partie entière.

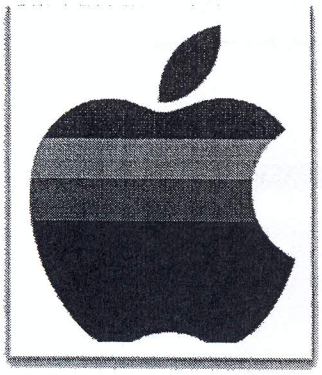

Image B (en niveau de gris)

Image C

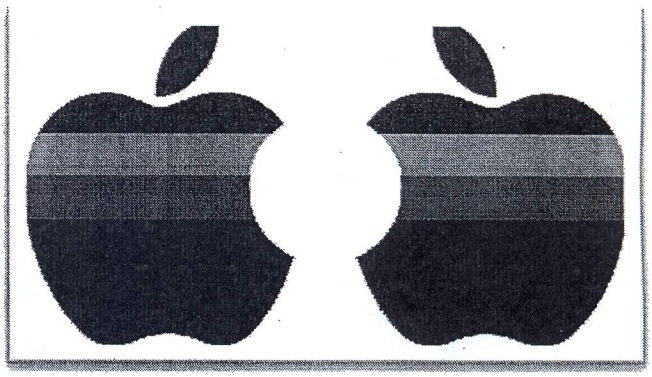

Image D

Dans la suite des questions on traitera des images en niveau de gris, décrite par une matrice dont chaque pixel est représenté seulement par un entier décrivant le ton de gris.

3. Ecrire une fonction **flipH**(img) qui renvoie la transformée de l'image img par la symétrie d'axe vertical. Par exemple, l'image C de la figure 2 résulte de l'application de ***flipH*** à l'image B.

4. Ecrire une fonction **poserH**(img1 , img2) qui prend en arguments deux images img1 et img2 de même hauteur et profondeur, et qui renvoie la nouvelle image obtenue en posant img1 à droite de img2 . Par exemple, l'image D de la résulte de l'application de ***poserH*** aux images B et C.

5. L'histogramme d'une image permet de compter le nombre de pixel d'un niveau de gris donné.

Écrire une fonction ***histo(im),*** qui prend en argument une image en niveau de gris (décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris). Cette fonction doit renvoyer une liste de taille 256 : en première position (indice 0), le nombre de pixels noirs (gris 0), en deuxième position (indice 1), le nombre de pixels gris 1, . . . , en dernière position (255), le nombre de pixels blancs (gris 255).

Annexes :

Importation du module :

import numpy as np

Création :

MonVecteur= *np.array([])*

Création automatique :

création d'une matrice nulle de 2 lignes et 3 colonnes : *np.zeros((2,3))*

création d'une copie : *MaCopie=np.copy(MonVecteur)*

Accession à un élément :

MaMatrice[1,2] #élément 2ème ligne, 3ème colonne (à partir de l'index 0)

accession à une ligne : MaMatrice[1,:] #2ème ligne

accession à une colonne : MaMatrice[:,2] #3ème colonne

modification d'un élément : MaMatrice[1,2]=10

Sélection d'une sous-matrice :

MaMatrice[0:2,0:2] # LigneDebut:LigneFin,ColonneDebut :ColonneFin (à partir de 0)

dimensions d'une matrice : np.shape(MaMatrice) #(NbreLignes,NbreColonnes)