Concours Blanc

Épreuve d'informatique

Proposition de corrigé

Exercice 1 : Traitement d'image

Une société de graphisme souhaite développer en interne une bibliothèque de fonctions python pour modifier des images aux formats standards.
Les images sont manipulées sous forme de tableau (`array`) à trois dimensions : ligne, colonne, couleur. Les couleurs étant, dans cet ordre, le rouge, le vert et le bleu.
Leur intensité est codée sur un octet comme un nombre entier compris entre 0 et 255.
Un pixel ayant la couleur `[0,0,0]` est noir, un pixel ayant la couleur `[255,255,255]` est blanc.
Certaines de ces fonctions ont déjà été écrites (*cf. annexe en dernière page*), d'autres sont à réaliser.

1. Justifier qu'un nombre entier codé sur un octet peut prendre des valeurs entières de 0 à 255.
Un octet est le regroupement de 8 bits, à savoir 8 chiffres en écriture binaire. Le nombre de combinaisons possibles est donc $2^8 = 256$, soit une pour chacun des 256 nombres entiers allant de 0 à 255.

2. Commenter la ligne 3 de l'entête.
La ligne 3 importe `mathplotlib.pyplot`, un composant de la bibliothèque `mathplotlib` qui permet des affichages à l'écran.
Ses instructions pourront être appelées à l'aide du préfixe `plt` comme dans l'instruction `plt.imshow(NomDeLImage)` aux lignes 18 et 43 de l'annexe.

3. Quelle est la différence entre le fichier `Images.png` et la variable `Image` ?
Le fichier `Images.png` est une image stockée dans l'ordinateur (dans le même dossier que celui où se trouve le programme Python) au format compressé `.png`.
La variable `Image` est un tableau de nombres (type `array`) manipulable par les instructions d'un programme Python.

4. Fonction `flou` : commenter les lignes 10 et 13 de la fonction `flou`
La ligne 10 lance la réalisation d'une boucle séquentielle, en donnant à la variable locale `j` les valeurs entières allant de 1 à $p-2$ qui correspondent au rang des colonnes de l'image à flouter.
La ligne 13 lance, à l'intérieur de trois autres boucles séquentielles (de variable respectivement `i`, `j` et `n`, la réalisation d'une boucle séquentielle, en donnant à la variable locale `m` successivement les valeurs -1, 0 et 1.
Ce nombre détermine le décalage de colonne dans la réalisation du floutage.

5. fonctions `conv` et `masque` : commenter les lignes 21, 26, 33, 34 et 39.

- Ligne 21 : Calcule la taille du masque, à savoir le nombre de pixels de côté du carré, et l'affecte à la variable `masque`.
- Ligne 26 : affecte à l'élément du masque de la ligne de rang n et de la colonne de rang m une valeur entrée par l'utilisateur via l'instruction `input()`.
  Remarque : Les lignes 25 et 26 ne peuvent pas être remplacées par la ligne :
  `masque[n,m]=input('ligne',n,'colonne',m,' : ')`
  car la fonction `input` *ne prend qu'un seul paramètre affichable.*

- Ligne 33 : en utilisant une instruction de la bibliothèque `numpy`, créée un tableau de zéros de même taille que le tableau `im`.
- Ligne 34 : créée une boucle séquentielle dont l'indice `i` est la ligne qui sera floutée. Elle démarrera à $i = x$ et se terminera à $i = n - x - 1$.
- Ligne 39 : ajoute à la couleur rouge du pixel `[i,j]` la valeur de la couleur rouge du pixel `[i+n,j+m]` multipliée par le coefficient de coordonnées `[n,m]` du masque `ma`.

6. Propositions de programmes
Choisir, en fonction du temps disponible et de ses affinités, parmi les propositions suivantes de réalisation de programme.
Codes dans le fichier CB image.py sur cahier de prépa.

a. Proposer une fonction `symetrie` qui renvoie le symétrique de l'image par rapport à son centre de symétrie.

```
def symetrie(im):
    n, p, s = im.shape
    sym = np.zeros_like(im)
    for i in range (n):
        for j in range(p):
            sym[i,j] = im[n-i-1,p-j-1]
    return plt.imshow(sym)
```

b. Proposer une fonction `rotation` qui réalise une rotation de l'image d'un quart de tour dans le sens des aiguilles d'une montre

```
def rotation(im):
    n, p, s = im.shape
    rot = np.zeros((p, n, s),dtype=np.uint8)
    for i in range(n):
        for j in range(p):
            rot[j, i]=im[n-i-1,j]
    return plt.imshow(rot)
```

c. On considère que la luminosité d'un pixel est la moyenne des nombres codant chacune de ses trois couleurs. Proposer une fonction `nb` qui permet l'affichage de l'image en noir et blanc (*chaque pixel est soit tout noir, soit tout blanc*) en fonction d'un seuil de luminosité. Cette fonction prend en paramètres l'image sous forme de tableau et un seuil de luminosité.

```
def nb(im,seuil):
    n, p, s = im.shape
    nb = np.zeros((n, p, s),dtype=np.uint8)
    # uint8 : Unsigned Integer coded with 8 bits
    for i in range(n):
        for j in range(p):
            lum=0
            for k in range(3):
                lum+=im[i,j,k]
            lum=int(lum/3)
            if lum>seuil:
                nb[i,j]=[255,255,255]
    return nb # plt.imshow(nb)
```

d. Le rendu de la fonction `nb` ci-dessus est incertain, en fonction de la luminosité de l'image initiale. Afin d'obtenir automatiquement un rendu plus satisfaisant, la société souhaite développer une fonction `nb2` qui appelle la fonction `nb` ci-dessus, et à l'aide d'une boucle conditionnelle, ajuste le seuil de luminosité de sorte qu'en sortie de boucle, l'image en noir et blanc ait « autant » de pixels noirs que de pixels blancs. La fonction `nb2` renverra en sortie l'image en noir et blanc ainsi que le seuil obtenu. Proposer une telle fonction.

```
def nb2(im):
    n, p, s = im.shape
    nbpix=n*p # nombre de pixels de l'image
    print(n,p,nbpix)
    seuil = 128 # seuil de passage au blanc
    nbpixnoir=0 # nombre de pixels noirs
    taux=nbpixnoir/nbpix
    while taux<0.49 or taux>0.51:
        imnb=nb(im,seuil)
        print('nbpixnoir,nbpix,taux,seuil',nbpixnoir,nbpix
   ,taux,seuil)
        input()
        nbpixnoir=0 # nombre de pixels noirs
        for i in range(n):
            for j in range(p):
                if imnb[i,j,0]==0:
                    nbpixnoir+=1
        taux=nbpixnoir/nbpix
        print(taux)
        input()
        if taux<0.5:
            seuil+=1
        else:
            seuil-=1
    return plt.imshow(imnb)
```

e. Proposer une fonction `etire` qui « étire » l'image d'un facteur 3 dans le sens « vertical »

```
def etire(im):
    n,p,s=im.shape
    etire = np.zeros((3*n,p,s),dtype=np.uint8)
    for i in range(n):
        for j in range(p):
            etire[3*i,j],etire[3*i+1,j],etire[3*i+2,j]=im[
  i,j],im[i,j],im[i,j]
    return plt.imshow(etire)
```

f. Proposer une fonction `PopArt` qui passe l'intensité de chacune des trois couleurs à 0 ou 255 selon que son nombre associé est, respectivement, inférieur ou supérieur à 128.

```
def PopArt(im):
    n, p, s = im.shape
    pa = np.zeros((n, p, s),dtype=np.uint8)
    # uint8 : Unsigned Integer coded with 8 bits
    for i in range(n):
        for j in range(p):
            for k in range(3):
                if im[i,j,k]>128:
                    pa[i,j,k]=255
    return plt.imshow(pa)
```

Annexe Exercice 1

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
Image=matplotlib.image.imread("Image.png")

def flou(im):
    n, p, s = im.shape
    flou= np.zeros_like(im)
    for i in range(1,n-1):
        for j in range(1,p-1):
            r,v,b=0,0,0
            for n in range(-1,2):
                for m in range(-1,2) :
                    r=r+im[i+n,j+m,0]
                    v=v+im[i+n,j+m,1]
                    b=b+im[i+n,j+m,2]
            flou[i, j] =[r/9,v/9,b/9]
    return plt.imshow(flou)

def masque(x):
    taille = 2*x+1
    masque=np.zeros((taille,taille))
    for n in range(taille):
        for m in range(taille):
            print('ligne',n,'colonne',m,' : ')
            masque[n,m]=input()
    return masque

def conv(im,x):
    n, p, s = im.shape
    ma=masque(x)
    print('Traitement en cours...')
    conv= np.zeros_like(im)
    for i in range(x,n-x):
        for j in range(x,p-x):
            r,v,b=0,0,0
            for n in range(-x,x+1):
                for m in range(-x,x+1) :
                    r=r+im[i+n,j+m,0]*ma[n,m]
                    v=v+im[i+n,j+m,1]*ma[n,m]
                    b=b+im[i+n,j+m,2]*ma[n,m]
            conv[i, j] =[r,v,b]
    return plt.imshow(conv)
```

Exercice 2 : Manipulation de chaînes de caractères, tri

Néo est un dissident dans un pays sous dictature.
Il souhaite échanger avec des correspondants via internet et sait que non seulement ses messages sont interceptés, mais qu'en outre les algorithmes informatiques de cryptage à sa disposition sont contrôlés par la dictature qui peut obtenir le message non crypté à partir du message qu'il aurait crypté par un tel procédé.

Il décide de créer lui même des algorithmes de cryptage de ses messages.

1. **Premier algorithme**
   Néo pense d'abord au code dit *« César »*, qui consiste à décaler toutes les lettres du texte d'un même déplacement dans l'alphabet.
   Par exemple si on déplace de +2 les lettres du texte `essai` on obtient `guuck`.

   a. Donner le texte crypté avec le code *César* du mot `bonjour` avec un déplacement de +3.
   `erqmrxu`

   b. *Pour enregistrer et manipuler du texte en informatique, on associe conventionnellement à 256 symboles usuels (caractères de l'alphabet en majuscule et en minuscule, virgule, point, espace, ...) un nombre entier unique compris entre 0 et 255 (on établit une bijection entre ces 256 symboles et l'ensemble* $[\![0, 255]\!]$*).*
   La fonction Python `ord()` prend comme argument un symbole usuel de texte et renvoie le nombre qui lui est associé. La fonction `chr()` réalise la bijection réciproque de `ord()`.
   Néo a réalisé la fonction `decaler()` donnée en annexe. Décrire le fonctionnement de cet algorithme.
   La variable locale `longueur` est affectée à la valeur de la longueur de la chaîne de caractères `texte`.
   La variable `crypt` est affectée à la valeur : chaîne de caractère vide.
   Ensuite les caractères sont isolés un à un dans le sens de lecture, leur code numérique est récupéré, et augmenté du décalage passé en paramètre de la fonction `decaler`.
   Enfin, au bout de la chaîne de caractère `crypt`, est ajouté le caractère correspondant au code augmenté du décalage.
   Après avoir ainsi traité tous les caractères, la chaîne de caractères cryptée est renvoyée par la fonction.

   c. Proposer un programme qui réalise la même fonction mais en utilisant l'instruction `while` plutôt que `for`, et qui teste la longueur de la chaîne `crypt`.

```
def decaler2(texte,decalage):
longueur=len(texte)
crypt=''
n=0
while len(crypt)<longueur:
    caractere=texte[n]
    code=ord(caractere)
    code=code+decalage
    crypt=crypt+chr(code)
    n=n+1
return(crypt)
```

d. Si la somme entre le nombre associé à un caractère et le déplacement du cryptage dépasse 255, le nombre obtenu pour le caractère codé ne correspond plus à un caractère. Dans ce cas, Néo décide d'enlever 256 au nombre obtenu. Proposer un ajout dans le code du programme initial pour traiter ce cas de figure.
(On écrira sur sa copie la ligne du programme précédent l'ajout pour repérer l'endroit de son insertion)

```
def decaler3(texte,decalage):
    longueur=len(texte)
    crypt=''
    for n in range(longueur):
        caractere=texte[n]
        code=ord(caractere)
        code=code+decalage
        if code>255:
            code=code-256
        crypt=crypt+chr(code)
    return(crypt)
```

2. Deuxième algorithme
Néo se dit que l'algorithme `decaler` est aisément déchiffrable. Il décide d'appliquer d'autres transformations à un texte à coder et crée la fonction `coupure()` donnée en annexe.

a. Donner le résultat renvoyé par l'instruction : `coupure('essai',3)`.
`aiess`

b. Que réalise cet algorithme ?

Cet algorithme est une fonction qui prend en variables une chaîne de caractères `texte` et un entier naturel `endroit`.
Elle affecte à la variable `premier_bout` la chaîne des caractères qui vont du début de `texte` à `endroit`, et à la variable `deuxieme_bout` la chaîne des caractères qui vont de `endroit` à la fin de la chaîne `texte`.
Enfin elle affecte à la variable `crypt` la concaténation des deux chaînes `deuxieme_bout` et `premier_bout` dans cet ordre.
Elle renvoie le texte où sont intervertis les deux parties du texte initial.

3. Troisième algorithme
Proposer un algorithme `combi` sous forme d'une fonction qui admet comme paramètre la chaîne `texte` à crypter et qui appelle les fonctions `decaler` et `coupure`.
On souhaite que cet algorithme `combi` réalise un cryptage difficile à déchiffrer. Cette fonction `combi` pourra prendre d'autres paramètres.

L'algorithme ci-dessous est une solution possible, qui utilise une boucle `for` pour combiner alternativement des décalages et des coupures avec des paramètres différents.

```
def combi(texte,n):
# on suppose que l'appelant de la fonction ne donnera pas
# un n plus grand que la longueur du texte
    for i in range(n):
        texte=decaler(texte,i)
        texte=coupure(texte,i)
    return(texte)
```

4. Quatrième et cinquième algorithme

Néo souhaite pouvoir déterminer si un caractère donné est présent dans un texte.
Pour cela il réalise deux fonctions Python.
La première s'appelle `tri()` et est donnée en annexe.

a. Décrire ce que fait la fonction `tri()`.

La fonction `tri` a une seule variable, la chaîne de caractère `texte`.
Elle parcourt la chaîne, et à chaque fois qu'elle rencontre un caractère dont le code associé par l'instrution `ord` est plus grand que celui du caractère suivant, elle permute les deux caractères.
Si elle a fait au moins une permutation après avoir ainsi parcouru tout le texte, elle recommence.
Quand elle parcourt le texte sans aucune permutation, alors elle renvoie la chaîne de caractère.

Cet algorithme, à l'aide d'une boucle séquentielle insérée dans une boucle conditionnelle, réalise au final le tri des lettres d'un texte dans l'ordre de leur code numérique associé croissant.

Remarque : Cette transformation de la chaîne de caractères n'est pas un cryptage car on ne peut retrouver le texte initial à partir du texte trié, le décryptage n'est pas possible, il y a perte d'information.
Elle pourrait cependant le devenir si on transmet également la succession des permutations réalisées.

b. La seconde fonction s'appelle `present()`, prend comme paramètres la chaîne `texte` renvoyée après la fonction `tri()` ainsi que le caractère à chercher. A l'aide d'un algorithme de dichotomie elle teste si le caractère est présent dans le texte soumis.
Proposer un code pour la fonction `present`.
Ci-dessus une solution.
Le choix des noms de variable et les commentaires visent à aider à la compréhension du script.

```
def present(texte_trie,caractere):
# Texte trié par la fonction tri, caractère dont on teste la présence
partie=texte_trie  # initialisation de la chaîne partie
compteur=1
while len(partie)>1:
    caractere_du_milieu=partie[len(partie)//2]
    if ord(caractere)<ord(caractere_du_milieu):
        # Si le code du caractère cherché est plus petit que le code du
        # caractère du milieu de la partie considérée
        partie=partie[:len(partie)//2]   # Garde la première partie du texte
    else:
        partie=partie[len(partie)//2:]   # Garde la seconde partie du texte
return(partie==caractere)   # variable booléenne
# A l'issue de la bouce while, il ne reste qu'un caractère qui est
# le caractère recherché si et seulement si celui-ci est présent.
```

Annexe Exercice 2

```
def decaler(texte,decalage):
    longueur=len(texte)
    crypt=''
    for n in range(longueur):
        caractere=texte[n]
        code=ord(caractere)
        code=code+decalage
        crypt=crypt+chr(code)
    return(crypt)

def coupure(texte,endroit):
    premier_bout=texte[:endroit]
    deuxieme_bout=texte[endroit:]
    crypt=deuxieme_bout+premier_bout
    return(crypt)

def tri(texte):
    longueur=len(texte)
    fini=False
    while fini==False:
        fini=True
        for n in range(longueur-1):
            caractere1=texte[n]
            caractere2=texte[n+1]
            if ord(caractere1)>ord(caractere2):
                fini=False
                texte1=texte[:n]
                texte2=texte[n+2:]
                texte=texte1+caractere2+caractere1+texte2
                print(texte)
    return(texte)
```