BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°34

DUREE DE L'EPREUVE : 1 heure

Le sujet comporte 3 pages numérotées de 1 / 3 à 3 / 3
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le candidat doit traiter les 2 exercices.

EXERCICE 1 (4 points)

Écrire une fonction `occurrence_max` prenant en paramètres une chaîne de caractères `chaine` et qui renvoie le caractère le plus fréquent de la chaîne. La chaine ne contient que des lettres en minuscules sans accent.

On pourra s'aider du tableau

```
alphabet=['a','b','c','d','e','f','g','h','i','j','k','l','m','n'
,'o,','p','q','r','s','t','u','v','w','x','y','z']
```

et du tableau `occurrence` de 26 éléments où l'on mettra dans `occurrence[i]` le nombre d'apparitions de `alphabet[i]`dans la chaine. Puis on calculera l'indice `k` d'un maximum du tableau `occurrence` et on affichera `alphabet[k]`.

Exemple :

```
>>> ch='je suis en terminale et je passe le bac et je souhaite
poursuivre des etudes pour devenir expert en informatique'
>>> occurrence_max(ch)
'e'
```

EXERCICE 2 (4 points)

On considère une image en 256 niveaux de gris que l'on représente par une grille de nombres, c'est-à-dire une liste composée de sous-listes toutes de longueurs identiques.

La largeur de l'image est donc la longueur d'une sous-liste et la hauteur de l'image est le nombre de sous-listes.

Chaque sous-liste représente une ligne de l'image et chaque élément des sous-listes est un entier compris entre 0 et 255, représentant l'intensité lumineuse du pixel.

Le négatif d'une image est l'image constituée des pixels `x_n` tels que `x_n + x_i = 255` où `x_i` est le pixel correspondant de l'image initiale.

Compléter le programme ci-dessous :

```
def nbLig(image):
    '''renvoie le nombre de lignes de l'image'''
    return ...

def nbCol(image):
    '''renvoie la largeur de l'image'''
    return ...

def negatif(image):
    '''renvoie le négatif de l'image sous la forme
       d'une liste de listes'''
```

```
    # on créé une image de 0 aux mêmes dimensions que le
paramètre image
    L = [[0 for k in range(nbCol(image))] for i in
range(nbLig(image))]

    for i in range(len(image)):
        for j in range(...):
            L[i][j] = ...
    return L

def binaire(image, seuil):
    '''renvoie une image binarisée de l'image sous la forme
       d'une liste de listes contenant des 0 si la valeur
       du pixel est strictement inférieure au seuil
       et 1 sinon'''

    # on crée une image de 0 aux mêmes dimensions que le
paramètre image
    L = [[0 for k in range(nbCol(image))] for i in
range(nbLig(image))]

    for i in range(len(image)):
        for j in range(...):
            if image[i][j] < ... :
                L[i][j] = ...
            else:
                L[i][j] = ...
    return L
```

Exemple :

```
>>> img=[[20, 34, 254, 145, 6], [23, 124, 237, 225, 69], [197,
174, 207, 25, 87], [255, 0, 24, 197, 189]]
>>> nbLig(img)
4
>>> nbCol(img)
5
>>> negatif(img)
[[235, 221, 1, 110, 249], [232, 131, 18, 30, 186], [58, 81, 48,
230, 168], [0, 255, 231, 58, 66]]
>>> binaire(img,120)
[[0, 0, 1, 1, 0], [0, 1, 1, 1, 0], [1, 1, 1, 0, 0], [1, 0, 0, 1,
1]]
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 1 du sujet 34 des épreuves pratiques NSI 2022

Remarques:
    * 'occurrence' = nombre d'apparition
    * cet exercice est DIFFICILE (trois boucles !) car on nous impose de
    travailler avec des tableaux. Après le code, j'ai ajouté la
    version simplifiée grâce aux dictionnaires !

    * on nous demande de travailler avec deux tableaux en parallèle
      et d'utiliser l'indice pour
       - dans le premier tableau : récupérer la lettre concernée
       - dans le deuxième tableau: récupérer le nombre d'apparition
    * pour parcourir `chaine`, deux façons :
       - pythonnesque: `for lettre in chaine:...`
       - classique:    `for i in range(len(chaine)):...

    * algo:
       - d'abord parcourir la chaine lettre par lettre
          - pour chaque lettre, parcourir tout le tableau de lettre
          pour idenfier l'indice
          - augmenter la valeur occurrence de l'indice trouvé
       - parcourir le tableau d'occurrence et identifier la valeur max
       - renvoyer la valeur max
"""



def occurrence_max(chaine):
    """
    Exemples et tests:
    >>> ch = 'je suis en terminale et je passe le bac et je souhaite poursuivre des
etudes pour devenir expert en informatique'
    >>> occurrence_max(ch)
    'e'
    """
    # initialisation du tableau contenant les lettres
    alphabet=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
              'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
              's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

    # BOUCLE 1 : parcours de la chaine lettre par lettre
    #  -> invariant : pour la zone parcourue jusqu'à présent,
    #  le tableau occurrence contient le nombre d'apparition
    #  de chaque lettre
    #
    #  -> initialement: rien n'a été parcouru donc 26 fois le `0`
    occurrence = [0] * 26

    #  -> condition d'arrêt: toute la chaine est parcourue
    for lettre in chaine:

        # BOUCLE 2: parcours du tableau alphabet
        #  -> invariant: la lettre a été cherchée dans alphabet[0 .. i-1]
        #  -> initialisation: i=0
        i = 0

        # condition d'arrêt :
        #  -> tout le tableau est parcouru: i == 26
        # OU
        #  -> la lettre est trouvée
        #
        # /!\ attention: si on inverse les conditions, il y a une erreur
        #     pourquoi...
        #     ...
        #     car si i vaut 26, alors alphabet[i] n'existe pas !
        #
        #     Mais si i vaut 26 ET que le test dans le bon sens,
        #     alors la condition s'arrête IMMÉDIATEMENT avant le OU
        #     (car i vaut 26)
        #     (et la suite n'est pas évaluée (heureusement ouf !))
        while not (i == 26 or alphabet[i] == lettre):
```

```
            i = i + 1

        # FIN BOUCLE 2
        # soit i vaut 26: la lettre n'est pas dans le tableau
        # soit i < 26: la lettre est dans le tableau
        #              et i est son indice (VICTOIRE !)

        if i < 26:
            # i est l'indice de la lettre courante
            # mise à jour du compteur d'occurrence de la lettre de la case i
            occurrence[i] = occurrence[i] + 1

    # BOUCLE 3: recherche de la valeur maximale du tableau occurrence
    #  -> invariant: i_max est l'indice de la valeur maxi de
    #                la zone occurrence[0 .. i-1]
    i_max = 0

    #   -> condition d'arrêt: après le tour de boucle i == 25
    for i in range(26):
        # mise à jour invariant si une valeur supérieure est trouvée
        if occurrence[i] > occurrence[i_max]:
            i_max = i

    # renvoie de la lettre d'indice i_max
    # et donc de la lettre avec la plus grande occurrence
    return alphabet[i_max]


ch = 'je suis en terminale et je passe le bac et je souhaite poursuivre des etudes p
our devenir expert en informatique'

# vérification avec une assertion
assert occurrence_max(ch) == 'e'

# vérification avec un affichage
print(occurrence_max(ch))

# vérification avec doctest
from doctest import testmod
testmod()



# pour information :
# version plus classique avec dictionnaire
#
# def occurrence_max_dic(chaine):
#     occurrence = {}
#     for lettre in chaine:
#         if lettre == ' ': continue
#
#         if lettre in occurrence:
#             occurrence[lettre] += 1
#         else:
#             occurrence[lettre] = 1
#
#     v_max = -1
#     for lettre in occurrence:
#         if occurrence[lettre] > v_max:
#             lettre_max = lettre
#             v_max = occurrence[lettre]
#
#     return lettre_max
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 2 du sujet 34 des épreuves pratiques NSI 2022

Remarques:
    * l'image est dans un tableau à deux dimensions : [[...], [...], ...]
    * c'est un tableau de tableaux
"""

def nbLig(image):
    '''renvoie le nombre de lignes de l'image'''
    # le nombre de ligne est égal au nombre de cases du tableau image
    return len(image)

def nbCol(image):
    '''renvoie la largeur de l'image'''
    # le nombre de colonne est égal au nombre de case de chaque élément
    # de l'image.
    # on prend ici la première case de image : image[0] et on calcule
    #  sa taille
    return len(image[0])

def negatif(image):
    '''renvoie le négatif de l'image sous la forme
       d'une liste de listes'''
    # on cree une image de 0 aux memes dimensions que le parametre image
    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))]

    # parcours de l'image où i est l'indice de la ligne courante
    for i in range(len(image)):
        # parcours de la ligne ou j est l'indice de la colonne courante
        for j in range(nbCol(image)):
            # l'énoncé dit que: pixel_negatif + pixel_normal = 255
            # ce qui se traduit par:   x_n + x_i = 255
            # on a donc l'affectation: x_n <- 255 - x_i
            L[i][j] = 255 - image[i][j]
    return L

def binaire(image, seuil):
    '''renvoie une image binarisée de l'image sous la forme
       d'une liste de listes contenant des 0 si la valeur
       du pixel est strictement inférieure au seuil
       et 1 sinon'''
    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))] # on crée un
e image de 0 aux mêmes dimensions que le paramètre image
    for i in range(len(image)):
        for j in range(nbCol(image)):

            # cas du pixel inférieur strictement au seuil
            if image[i][j] < seuil :
                L[i][j] = 0
            # cas du pixel supérieur ou égal au seuil
            else:
                L[i][j] = 1
    return L


img=[[20, 34, 254, 145, 6], [23, 124, 287, 225, 69], [197, 174, 207, 25, 87], [255,
0, 24, 197, 189]]

# vérification avec des assertions
assert nbLig(img) == 4
assert nbCol(img) == 5
assert negatif(img) == [[235, 221, 1, 110, 249], [232, 131, -32, 30, 186], [58, 81,
48, 230, 168], [0, 255, 231, 58, 66]]
assert binaire(img,120) == [[0, 0, 1, 1, 0], [0, 1, 1, 1, 0], [1, 1, 1, 0, 0], [1, 0
, 0, 1, 1]]

# vérification avec des affichages
print(nbLig(img))
print(nbCol(img))
print(negatif(img))
print(binaire(img,120))
```