info201 : algorithmique et programmation
Examen
CORRECTION

Pierre Hyvernat
Ilham Alloui
UFR Sciences Fondamentales et Appliquées, université de Savoie
Pierre.Hyvernat@univ-smb.fr
Ilham.Alloui@univ-smb.fr

Documents et calculatrices interdits.

Durée : 1h30.

Un barème provisoire est donné dans la marge, un point négatif est réservé pour la présentation.

Rappel : vous pouvez utiliser les fonctions demandées pour écrire celles des questions suivantes, *même si vous ne les avez pas écrites.*

Partie 1 : tableaux et boucles

(1) *Question 1.* Écrivez une fonction qui calcule la somme de tous les éléments d'un tableau à une dimension :

```
>>> somme1([1,2,3,4])
10
```

Solution :

```
def somme1(T):
    r = 0
    for n in T:
        r = r + n
    return r
```

(2) *Question 2.* Écrivez une fonction qui calcule la somme de tous les éléments d'un tableau à deux dimensions :

```
>>> somme2([[16],[3,-2,2000,0],[]])
2017
```

Solution :

```
def somme2(T):
    r = 0
    for t in T:
      r = r + somme1(t)
    return r
```

(4) *Question 3.* Une *matrice* est un tableau à deux dimensions (un "tableau de lignes") qui satisfait les conditions suivantes :

- il contient au moins une ligne et une colonne,
- chaque ligne contient le même nombre de cases.

Parmi les définitions suivantes, seule `M` est une matrice.

```
>>> T1 = []
>>> T2 = [[]]
```

```
>>> D  = [  [1, 2, 3],
            [4, 5],
            [6]        ]
>>> M  = [ ['a', 'b', 'c'],
           ['d', 'e', 'f'],
           ['g', 'h', 'i'],
           ['j', 'k', 'l'] ]
```

Écrivez une fonction `est_matrice` qui renvoie `True` lorsque son argument est une matrice, et `False` sinon.

Note : 1 point du barème est réservé au fait que votre fonction s'arrête dès que possible.

Solution :

```
def est_matrice(T):
    if len(T) == 0:
        r = False
    elif len(T[0]) == 0:
        r = False
    else:
        l = len(T[0])
        i = 1
        r = True
        while i<len(T) and r:
            if len(T[i]) != l:
                r = False
            i = i+1
    return r
```

Partie 2 : chaines de caractères

On rappelle que la méthode `append` permet de rajouter une case à la fin d'un tableau de manière efficace.

(2) *Question 1.* On dispose d'un tableau de chaines de caractères que l'on souhaite "coller" en les séparant par des espaces pour obtenir une unique chaine. Écrivez la fonction `colle_mots` correspondante :

```
>>> colle_mots(["Vive", "les", "équations", "du", "second", "degré", "!!!"])
'Vive les équations du second degré !!!'
>>> colle_mots(["Atchoum"])
'Atchoum'
>>> colle_mots([])
''
```

Note : vous n'avez pas le droit d'utiliser la méthode `.join` vue en cours.

Solution :

```
def colle_mots(T):
    if len(T) == 0:
        r = ""
    else:
        r = T[0]
        for i in range(1, len(T)):
            r = r + " " + T[i]
    return r
```

(2) *Question 2.* On dispose d'une liste de "lignes", où chaque "ligne" est elle-même une liste de mots (chaine de caractères). Écrivez une procédure `affiche_lignes` qui affiche les "lignes" correctement :

- les mots à l'intérieur de chaque ligne sont séparés par des espaces,
- les lignes sont séparées par un saut de ligne simple.

Par exemple :

```
>>> affiche_lignes([ ["Vive", "les", "équations", "du", "second"],
                     ["degré", "!!!"] ])
Vive les équations du second
degré !!!
```

Solution :

```
def affiche_lignes(T):
    for ligne in T:
        print(colle(ligne))
```

(4) *Question 3.* Les logiciels de traitement de texte vont automatiquement à la ligne lorsque les mots ne tiennent pas sur la largeur de la page. L'objectif de cette question est d'écrire une fonction `coupe` pour décider où placer les sauts de ligne entre les mots d'un paragraphe.

Cette fonction prend deux arguments :

- une liste de mots (des chaines de caractères) : `mots`,
- une largeur maximale (nombre entier) : `largeur`.

Son résultat sera un tableau à deux dimensions : chaque élément du résultat est une liste de mots (chaines de caractères) representant une ligne de texte.

Par exemple :

```
>>> coupe(["Vive", "les", "équations", "du", "second", "degré", "!!!"], 40)
[["Vive", "les", "équations", "du", "second", "degré", "!!!"]]

>>> coupe(["Vive", "les", "équations", "du", "second", "degré", "!!!"], 30)
[["Vive", "les", "équations", "du", "second"], ["degré", "!!!"]]

>>> coupe(["Vive", "les", "équations", "du", "second", "degré", "!!!"], 20)
[["Vive", "les", "équations"] , ["du", "second", "degré", "!!!"]]

>>> coupe(["Vive", "les", "équations", "du", "second", "degré", "!!!"], 10)
[["Vive", "les"], ["équations"] , ["du", "second"], ["degré", "!!!"]]
```

Le résultat du 2ème exemple s'explique par

```
<------  30 caractères   ------>
|------------------------------|
|                              |
|Vive les équations du second  |
|degré !!!                     |
|                              |
|------------------------------|
```

Il n'est en effet pas possible de rajouter le mot `"degré"` à la fin de la première ligne sans dépasser la limite de 30 caractères.

Indication : votre fonction pourra conserver une liste de mots pour représenter une ligne de texte, ainsi que sa taille. Elle devra essayer d'y ajouter les mots un par un. Lorsqu'un mot rendrait cette liste trop grande :

- la liste représente une ligne de texte complète et peut être ajoutée au résultat,
- le mot en question commence une nouvelle ligne de texte et il faut réinitialiser cette liste avec ce mot seul.

Attention :

- il ne faut pas oublier de compter les espaces ajoutés entre chaque mot dans la taille de la ligne de texte courante,
- il ne faut pas oublier la dernière ligne dans le résultat final.

Solution :

```
def coupe(mots, largeur):
    resultat = []
    ligne_courante = []
    taille_courante = 0
    for i in range(0, len(mots)):
        mot = mots[i]
        if taille_courante + 1 + len(mot) <= largeur:
            ligne_courante.append(mot)
            taille_courante = taille_courante + 1 + len(mot)
        else:
            resultat.append(ligne_courante)
            ligne_courante = [mot]
            taille_courante = len(mot)
    if ligne_courante != []:
        resultat.append(ligne_courante)
    return resultat
```

(2) *Question 4.* On suppose qu'on dispose d'une fonction `decoupe_mots` qui transforme une chaine de caractères en liste de mots :

```
>>> decoupe_mots("Vive les équations du second degré !!!")
["Vive", "les", "équations", "du", "second", "degré", "!!!"]
```

Écrivez la fonction finale `formater` qui prend en argument une chaine de caractères et une largeur, et affiche cette chaine en ajoutant les sauts de lignes aux endroits appropriés :

```
>>> formater("Vive les équations du second degré !!!", 40)
Vive les équations du second degré !!!

>>> formater("Vive les équations du second degré !!!", 30)
Vive les équations du second
degré !!!

>>> formater("Vive les équations du second degré !!!", 20)
Vive les équations
du second degré !!!

>>> formater("Vive les équations du second degré !!!", 10)
Vive les
équations
du second
degré !!!
```

Solution :

```
def formater(chaine, largeur):
    mots = decoupe_mots(chaine)
    lignes = coupe(mots, largeur)
    affiche_lignes(lignes)
```

Partie 3 : fichiers

On rappelle que les fonctions relatives aux fichiers sont les suivantes :
- ouverture d'un fichier en lecture : `f = open(..., "r")`
- ouverture d'un fichier en écriture : `f = open(..., "w")`
- fermeture d'un fichier : `f.close()`
- écriture d'une chaine dans un fichier : `f.write(...)`
- lecture d'une ligne dans un fichier : `l = f.readline()`.

Aucune autre fonction ou méthode sur les fichiers n'est autorisée.

(3) *Question 1.* Écrivez une fonction qui analyse les lignes d'un fichier et calcule en même temps :
- la taille maximale des lignes,
- la taille minimale des lignes,
- la moyenne des tailles des lignes.

Le résultat de la fonction sera un dictionnaire à trois cases contenant les valeurs calculées.

Notes :
- on suppose que toutes les lignes sont terminées par un saut de ligne, qu'il ne faut pas compter dans les tailles de lignes,
- vous ne devez ouvrir le fichier qu'une seule fois...

Solution :

```
def analyse_lignes(nom_fichier):
    f = open(nom_fichier, mode='r')
    ligne = f.readline()
    taille_min = len(ligne)-1
    taille_max = len(ligne)-1
    taille_totale = len(ligne)-1
    nb_lignes = 0

    while ligne != "":
        l = len(ligne) - 1
        if l < taille_min:
            taille_min = l
        if l > taille_max:
            taille_max = l
        taille_totale = taille_totale + l
        nb_lignes = nb_lignes + 1
        ligne = f.readline()

    f.close()

    return { "min":     taille_min,
             "max":     taille_max,
             "moyenne": taille_totale / nb_lignes }
```