INFORMATIQUE : CONCOURS BLANC

03 MAI 2017

Traumathisés par la MPSI, des étudiants décident de partir loin dans l'espace pour refaire leur vie. Ils explorent des planètes capables de les accueillir. Ils pensent avoir trouvé leur bonheur du coté de Proxima du SansTort. Ils ont héroïquement récupéré des échantillons pour analyser l'atmosphère. Les données vont être compliquées à analyser, heureusement pour eux, ils ont apporté leurs connaissances en informatique.
Dans l'espace, personne ne vous entendra coder...

PARTIE A - PRÉLIMINAIRES

Afin de se préparer au mieux leur étude des échantillons de l'atmosphère, les étudiants ont besoin de quelques fonctions afin de ne pas avoir de doublons dans leurs données.

1. Écrire une fonction `del_double(L)` qui prends en argument une liste triée L et renvoie une liste contenant les éléments de L en ayant éliminé les éléments apparaissant plusieurs fois. La liste finale doit toujours être triée. On s'assurera que la complexité est linéaire en la taille de L. Par exemple :

   ```
   >>> del_double([1,1,3,3,3,7])
   >>> [1,3,7]
   ```

2. Proposer un exemple de liste non triée sur lequel la fonction `del_double(L)` ne renvoi pas le résultat voulu.
3. Écrire une fonction `egal(L1,L2)` renvoyant un booléen permettant de savoir si deux listes $L1$ et $L2$ sont identiques.
4. Étudier la complexité dans le pire des cas de cette fonction.

Des données collectés sont transmises sous la forme de couples `(tab,chk)` ou `tab` est un tableau python de valeurs entières et `chk` est une valeur qui correspond à la somme des valeurs absolues de `tab`, on parle de checksum. Le but d'un checksum est de vérifier que les données ont été transmises sans erreurs.

5. Ecire une fonction `checksum(tab,chk)` qui renvoi un booléen selon que la somme des valeurs absolues du tableau `tab` soit égale à `chk`

   ```
   >>> checksum ( [3,-5,7],15)
   >>> True
   ```

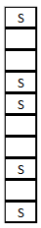
PARTIE B - ANALYSEUR D'ATMOSPHÈRE

Ils disposent d'un analyseur pour leur échantillons. Cette machine a la forme d'une colonne découpée en plusieurs compartiments. On dépose un échantillon dans le compartiment du haut, puis une fois analysé il descend dans le compartiment suivant. On peut alors insérer un nouvel échantillon dans le compartiment du haut, chacun travaillant indépendamment des autres. Les compartiments sont au nombre de n numérotés de 0 à $n-1$ en partant du bas.

Pour modéliser cette machine, on construit une liste L contenant uniquement des 0 ou des 1. L'élément en position k vaut 1 si un échantillon est présent au compartiment k en partant du bas.

Ainsi la liste L simule la situation suivante :

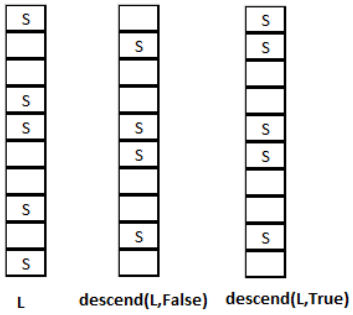
```
L = [1,0,1,0,0,1,1,0,0,1]
```



1. Écrire une fonction `nb_echant(L)` qui renvoi le nombre d'échantillons présents dans la machine.

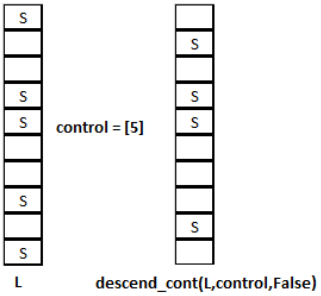
2. Écrire une fonction `cohérence(L)` qui renvoi un booléen selon que la liste ne contiennent que des 0 et des 1 ou pas.

A la fin des analyses de chaque compartiment les échantillons descendent dans le compartiment suivant (celui d'en dessous). L'élément dans le compartiment du bas sort de la machine : il n'est plus dans la liste, et on peut choisir de rajouter un élément dans le compartiment du haut : en fin de liste.
On souhaite construire une fonction `descend(L,b)` qui simule une étape de descente pour la liste L avec b un booléen qui vaut `True` si on ajoute un élément dans le compartiment du haut, et `False` sinon. Cette fonction renvoi la liste L mise à jour.
Un exemple d'évolution :



3. Que doit renvoyer l'appel `descend([1,0,1,1,0,1],True)` ?
4. Écrire cette. fonction `descend(L,b)`.

Il peut arriver qu'un échantillon doive subir un autre test sur le même compartiment. Ce compartiment reste bloqué et l'échantillon ne descends pas. On dispose d'une liste `control` qui contient les numéros des compartiments qui sont bloqués. La descente doit respecter cette contrainte et celles qu'elle induit : un échantillon qui ne se déplace pas peut en bloquer d'autres. On souhaite construire une fonction `descend_cont(L,control,b)` qui réalise cela.
Un exemple :



5. Écrire cette fonction `descend_cont(L,control,b)`
6. La machine se met en veille si il n'y a plus d'échantillons en analyse. Écrire une condition booléenne sur L qui traduit cela.

PARTIE C - ANALYSE DES DONNÉES

Les données analysées sont récupérées par la machine et transmise via des fichiers .txt.
En fouillant leur archives les élèves tombent sur un vieux TP d'extraction de données qui contient la fonction suivante :

```
def lecture_fichier(fichier):
    f = open(fichier,mode='r')
    temps,concen = [ ],[ ]
    for ligne in f:
        t,c = ligne.split(';')
        temps.append(float(t))
        concen.append(float(c))
    f.close()
    return temps,concen
```

Ils l'utilisaient sur un fichier issu de latis pro d'une expérience de chimie pour récupérer le temps et la concentration d'un produit. De mémoire le fichier avait la tête suivante :

```
1;0.01
1.5;0.05
1.7;0.13
...
```

Hélas la machine fourni trois données, le temps, le taux d'oxygène et le taux de radiations, de plus elle inscrit un en-tête dans chaque fichier et la séparation est un ' :'.

```
# temps,oxy,rad
0.2 : 0.10 : 0.02
0.5 : 0.13 : 0.04
1.5 : 0.08 : 0.03
...
```

1. Proposer une fonction `lecture_fichier` modifiée qui fonctionne pour les données transmises par la machine et renvoi trois listes `temps,oxygene,radiation`.

On suppose que maintenant les trois listes `temps,oxygene,radiation` sont accessibles dans le code.

2. Le taux de radiation peut être extrêmement dangereux sur la longueur. Il faut donc calculer l'intégrale du taux de radiation sur le temps écoulé. Proposer un code qui détermine une valeur approchée `radiatot` de cet intégrale via la méthode des rectangles en utilisant les listes `temps` et `radiations`.
3. Une variation trop brusque du taux d'oxygène est problématique pour la survie. Pour s'en assurer il faut calculer $\sum_{i=0}^{n-1} |ox_{i+1} - ox_i|$ où ox_i représente l'élément en position i de la liste `oxygene` et n la longueur de cette liste. Proposer un code qui calcule cette somme `var_oxy`.
4. Un test est concluant si `radiatot` est inférieur à 10 et si `var_oxy` est infieur à 20. Écrire une condition booléenne qui traduit ces deux conditions.

PARTIE D - ÉTUDE DE VIABILITÉ

Pour répondre à un problème de viabilité il est nécessaire de résoudre une équation différentielle du second ordre de la forme $y'' = f(y)$ dont les conditions initiales sont $y(0) = 10$ et $y'(0) = -0.02$
On considère que la fonction numérique f est saisie dans le code Python.

1. Afin de résoudre numériquement on pose $z = y'$. Écrire le problème sous forme de système différentiel du premier ordre pour y et z.

Avec la méthode d'Euler explicite, pour un pas h on obtient deux suites (y_i) et (z_i) approchant les valeurs de y et z par

$$y_{i+1} = y_i + hz_i \text{ et } z_{i+1} = z_i + hf(y_i)$$

2. Compléter le code suivant qui permet d'obtenir les listes Ly et Lz contenant les valeurs de (y_i) et (z_i) :

```
n = 10000
h = 0.002
y,z = ??? # valeurs initiales
Ly,Lz = ???? #  listes de stockage
for in range(n-1) :
    fy = f(y)
    y = ???
    z = ???
    Ly.append(y)
    Lz.append(z)
```

La méthode d'Euler n'étant pas précise, les élèves décident de passer au schéma de Verlet, qui est donné par :

$$y_{i+1} = y_i + hz_i + \frac{h^2}{2} f(y_i) \text{ et } z_{i+1} = z_i + \frac{h}{2}\left(f(y_i) + f(y_{i+1})\right)$$

3. Proposer un code inspiré de celui d'Euler qui d'obtenir les listes Ly et Lz contenant les valeurs de (y_i) et (z_i) pour le schéma de Verlet.