Devoir Surveillé n°3 d'Informatique

Samedi 13 Janvier 2024 *(Durée : 2 heures)*

Exercice 1

Le but de cet exercice est d'écrire un programme qui, partant d'une durée donnée en secondes, l'exprime en jours, heures, minutes et secondes.

Exemple : 95 381 secondes correspondent à 1 jour 2 heures 29 minutes 41 secondes.

1) Que fait le programme suivant ?

```
d=eval(input('durée en secondes : '))
m=d//60
s=d%60
```

2) Écrire alors le programme complet qui demande une durée en secondes et l'affiche en jours, heures, minutes et secondes

Exercice 2

On considère la fonction `Python` suivante (qui s'applique à une liste non vide de réels) :

```
def maxi(L):
    m=L.pop()
    while len(L)>0:
        if m<L.pop():
            m=L.pop()
    return m
```

1) Que renvoient les instructions suivantes ? Vous justifierez **précisément** vos réponses.

   (a) `maxi([1,3,2])` (b) `maxi([1,2])` (c) `maxi([1,3,2,4])` (d) `maxi([2,1])`

   Attention à bien simuler l'exécution de la fonction pas à pas, ce n'est pas si simple ... il y a des problèmes et on n'obtient pas forcément ce à quoi on s'attend

2) On aimerait que l'instruction `maxi(L)` renvoie le maximum de la liste `L`, quelle que soit la taille de cette dernière. Modifier la fonction `maxi(L)` en conséquence.

   Il s'agit simplement de corriger ce qui cloche ici, pas de changer la méthode de calcul !

Exercice 3

Écrire une fonction `Somme_Inverses(n)` qui renvoie la somme $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$.

Remarque : *bien entendu, cette fonction ne s'appliquera qu'à des entiers strictement positifs ... mais c'est à l'utilisateur de faire attention ! Pas besoin de vous embêter à tester la conformité de la variable dans le corps de la fonction, vous partez du principe que n appartient à* $\mathbb{N}^*$.

Exercice 4

On dispose en mémoire d'une variable entière $p > 0$.
On exécute alors le programme suivant :

```
c=0
while p>0:
    if c==0:
        p=p-2
        c=1
    else:
        p=p+1
        c=0
```

1) Dressez le tableau d'exécution de ce programme dans le cas où $p = 5$.
2) On désire montrer que la boucle `while` se termine toujours.
   Pour cela, prouvez que $v = 2p + 3c$ est un variant de boucle.

Exercice 5

On définit la suite de Héron [1] par $u_0 = 1$ et $u_{n+1} = \frac{u_n}{2} + \frac{1}{u_n}$ pour $n \in \mathbb{N}$.

1) Écrire une fonction `SuiteU(n)` qui renvoie la liste $[u_0, u_1, \ldots, u_n]$.
2) Comment représenter graphiquement les 20 premiers termes de cette suite $(u_n)_{n\in\mathbb{N}}$?
3) Écrire une fonction `MoyenneU(n)` qui renvoie la quantité $\frac{u_0+u_1+\cdots+u_n}{n+1}$.

Exercice 6

1) Écrire *en français* un algorithme qui calcule le produit des entiers de 1 à 99.
2) Traduire cet algorithme en langage `Python`.

1. Héron d'Alexandrie, mathématicien grec contemporain de Pline l'Ancien ... qui n'a évidemment aucun rapport avec les petits patapons.

Exercice 7

Une des méthodes utilisées pour trouver le minimum d'une fonction numérique f est l'algorithme de Nelder-Mead. C'est une procédure itérative simple : on part d'un segment initial [MN], qui se déplace et se réduit jusqu'à ce que ses extrémités se rapprochent d'un point où la fonction présente un minimum (qui peut être local et non global).

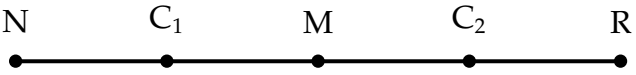
Soient x_M et x_N les abscisses des points M et N. L'algorithme se décompose en trois étapes.

❶ La première étape consiste à réindexer si nécessaire les deux extrémités du segment (càd échanger M et N) de manière à ce que $f(x_M) \leqslant f(x_N)$.

❷ L'extrémité N, pour laquelle f est maximale, est remplacée par une nouvelle extrémité. On définit pour cela le point R, symétrique de N par rapport à M, par $x_R = 2\,x_M - x_N$.

- Si $f(x_R) < f(x_M)$, on remplace N par R.
- Sinon, on définit les milieux C_1 et C_2 des segments [MN] et [MR] respectivement par $x_{C_1} = \frac{1}{2}(x_M + x_N)$ et $x_{C_2} = \frac{1}{2}(x_M + x_R)$.
  - Si $f(x_{C_1}) < f(x_M)$, on remplace N par C_1.
  - Sinon, on remplace N par C_2.

❸ On obtient ainsi un nouveau segment [MN]. Il n'y a plus qu'à réitérer le procédé.
Pour éviter les boucles infinies, on fixera une nombre maximal d'itérations `Itmax`.



Soient `f` *une fonction dont on recherche le minimum,* `xM` *et* `xN` *les abscisses des extrémités du segment initial* `[MN]`*. Nous allons écrire des fonctions réalisant les trois étapes de l'algorithme.*

1) Écrire une fonction `reindexation(f,xM,xN)` qui renvoie les extrémités `(xM,xN)` du segment `[MN]` après réindexation (première étape).

2) Écrire une fonction `iteration(f,xM,xN)` qui, partant d'un segment `[MN]` réindexé, renvoie ses extrémités `(xM,xN)` à l'issue d'une itération de la méthode de Nelder-Mead (deuxième étape).

Il reste à réitérer la méthode Nelder-Mead jusqu'à ce que l'écart relatif entre x_M et x_N, défini par $\delta(x_M, x_N) = \dfrac{2\,|x_M - x_N|}{|x_M| + |x_N|}$*, soit inférieur à un seuil fixé par avance.*

3) Écrire une fonction `ecart(xM,xN)` qui renvoie l'écart relatif $\delta(x_M, x_N)$ entre x_M et x_N.

4) Écrire une fonction `boucle(f,a,b,Itmax,eps)` qui réalise des itérations successives de la méthode de Nelder-Mead en partant des valeurs initiales $x_M = a$ et $x_N = b$, jusqu'à ce que l'écart relatif entre x_M et x_N soit inférieur à `eps` ou que l'on atteigne le nombre maximum d'itérations `Itmax`. La fonction renvoie alors le couple (x_M, x_N).

Les curieux pourront tester cet algorithme sur des fonctions simples chez eux ...