Examen Programmation ENSAE première année 2007 (rattrapage)
Examen oral (30 minutes)

0.0.1 Position initiale d'un élément après un tri **

Enoncé

On suppose qu'on peut trier la liste `l` = [...] avec la méthode `sort`. Une fois le tableau trié, comment obtenir la position du plus petit élément dans le tableau initial ?

Correction

L'idée est de ne pas trier la liste mais une liste de couples incluant chaque élément avec sa position. On suppose que la liste `l` existe. La méthode `sort` utilisera le premier élément de chaque couple pour tier la liste de couples.

```
l2 = [ ( l [i], i ) for i in range (0, len (l)) ]
l2.sort ()
print l2 [0][1]  # affiche la position du plus petit élément
                 # dans le tableau initial
```

fin exo 0.0.1 □

0.0.2 Comprendre une erreur de logique *

Enoncé

1) Quel est le résultat affiché par le programme suivant :

```
def ensemble_lettre (s) :
          ens = []
          for i in range (0, len (s)) :
              c = s [i]
              if c in ens :
                  ens.append (c)
          return ens

print lettre ("baaa")
```

Est-ce que ce résultat change si on appelle la fonction `ensemble_lettre` avec un autre mot ?

2) Le programme précédent n'est vraisemblablement pas fidèle aux intentions de son auteur. Celui-ci avait pour objectif de déterminer l'ensemble des lettres différentes de la chaîne de caractères passée en entrée. Que faut-il faire pour le corriger ? Que sera le résultat de l'instruction `ensemble_lettre`("`baaa`") en tenant compte de la modification suggérée ?

Correction

1) L'instruction `if c in ens :` signifie que le caractère `c` est ajouté seulement s'il est déjà présent dans la liste `s` qui est vide au début de la fonction. Elle sera donc toujours vide à la fin de l'exécution de la fonction et sera vide quelque soit le mot `s` fourni en entrée comme paramètre. Le résultat ne dépend donc pas du mot.

2) Il suffit de changer `if c in ens :` en `if c not in ens :` pour donner :

```
def ensemble_lettre (s) :
    ens = []
    for i in range (0, len (s)) :
        c = s [i]
        if c not in ens :
            ens.append (c)
    return ens
```

Le résultat pour le mot "`baaa`" est ["`b`","`a`"].

fin exo 0.0.2 □

0.0.3 Comprendre une erreur d'exécution *

Enoncé

```
k = [10,14,15,-1,6]
l = []
for i in range (0,len (k)) :
    l.append ( k [ len (k) - i ] )
```

Le programme génère une exception de type `IndexError`, pourquoi ?

Correction

Lors du premier passage dans la boucle `for`, le premier ajouté dans la liste `l` est `k[len(k)]` qui n'existent pas puisque les indices vont de 0 à `len(k)`. Voici la correction :

```
k = [10,14,15,-1,6]
l = []
for i in range (0,len (k)) :
    l.append ( k [ len (k) - i-1 ] )  # -1 a été ajouté
```

fin exo 0.0.3 □

0.0.4 Précision des calculs ***

Enoncé

On cherche à calculer $\ln 2$ grâce à la formule $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + ...$ Pour $x = 1$, cette suite est convergente et permet de calculer $\ln 2$.

1) On propose deux programmes. Le premier utilise une fonction `puiss`. Le second ne l'utilise pas :

```
def puiss (x, n) :
    s = 1.0
    for i in range (0,n) :
        s *= x
    return s

def log_suite (x) :
    x    = float (x-1)
    # ......
    s    = 0
    old  = -1
    n    = 1
    while abs (old - s) > 1e-10 :
        old = s
        po  = puiss (x,n) / n
        if n % 2 == 0 : po = -po
        s  += po
        n  += 1
        # ......
    return s

print log_suite (2)
```

```
# ......
# ......
# ......
# ......
# ......
# ......
def log_suite (x) :
    x    = float (x-1)
    x0   = x
    s    = 0
    old  = -1
    n    = 1
    while abs (old - s) > 1e-10 :
        old = s
        po  = x / n
        if n % 2 == 0 : po = -po
        s  += po
        n  += 1
        x  *= x0
    return s

print log_suite (2)
```

Quel est le programme le plus rapide et pourquoi ? Quels sont les coûts des deux algorithmes ?

2) Combien faut-il d'itérations pour que la fonction retourne un résultat ?

3) On introduit la fonction `racine_carree` qui calcule $\sqrt{k}$. Celle-ci est issue de la résolution de l'équation $f(x) = 0$ avec $f(x) = x^2 - k$ à l'aide de la méthode de *Newton*[1].

```
def racine_carree (k) :
    x0 = float (k)+1
    x  = float (k)
    while abs (x-x0) > 1e-10 :
        x0 = x
        x  = (k-x*x) / (2 * x) + x
    return x
```

Cette fonction retourne un résultat pour $\sqrt{2}$ en 6 itérations. On décompose ensuite $\ln 2 = \ln\left((\sqrt{2})^2\right) = 2\ln\sqrt{2} = 2\ln\left(1+\left(\sqrt{2}-1\right)\right)$. En utilisant cette astuce et la fonction `racine2`, à combien estimez-vous grossièrement le nombre d'itérations nécessaires pour calculer $\ln 2$: 10, 40 ou 100 ? Justifiez. On rappelle que $2^{10n} \sim 10^{3n}$.

4) Que proposez-vous pour calculer $\ln 100$?

Correction

1) Le second programme est le plus rapide. A chaque boucle du premier programme, la fonction `puiss` calcule x^n avec n multiplications, ce calcul est remplacé par une seule multiplication dans le second programme.

Les deux programmes exécuteront exactement le même nombre de fois la boucle `while`. Soit n ce nombre d'itérations, le coût du premier programme est $1 + 2 + 3 + 4 + ... + n \sim O(n^2)$ car la fonction `puiss` fait i passages dans sa boucle `for` pour l'itération i. Le coût du second programme est en $O(n)$.

2) Si on pose $s_n = \sum_{k=1}^{n} (-1)^{k+1} \frac{x^k}{k}$, la condition d'arrêt de la fonction `log_suite` correspond à :

$$|s_n - s_{n-1}| > 10^{-10} \iff \left|\frac{x^k}{k}\right| > 10^{-10}$$

1. Cette méthode est utilisée pour résoudre l'équation $f(x) = 0$. On construit une suite convergeant vers la solution définie par $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. Sous certaines conditions, la suite (x_n) converge vers la solution. Il suffit d'appliquer cela à la fonction $f(x) = x^2 - k$ où k est le nombre dont on veut trouver la racine.

Comme on calcule $\ln(1+1)$, $x=1$, la condition est donc équivalente à :

$$\frac{1}{k} > 10^{-10} \iff k < 10^{10}$$

La fonction `log_suite` fait dix milliards de passages dans la boucle `while`, dix milliards d'itérations.

3) Cette fois-ci, on cherche à calculer $\ln\sqrt{2} = \ln\left(1+\sqrt{2}-1\right)$, d'où $x = \sqrt{2}-1$. La condition d'arrêt est toujours $\left|\frac{x^k}{k}\right| > 10^{-10}$ et on cherche à majorer le nombre d'itérations nécessaires pour que la fonction converge vers un résultat. $\sqrt{2} \sim 1.414 < \frac{1}{2}$. On majore donc x par $\frac{1}{2}$:

$$\begin{aligned}
&\implies \left|\frac{\left(\sqrt{2}-1\right)^k}{k}\right| > 10^{-10} \\
&\implies \left|\frac{1}{2^k k}\right| > 10^{-10} \\
&\implies \frac{1}{2^k} > 10^{-10} \implies 2^k < 10^{10}
\end{aligned}$$

On utilise l'approximation donnée par l'énoncé à savoir $2^{10n} \sim 10^{3n}$.

$$2^k < 10^{10} \iff 2^{10k} < 10^{100} \sim 10^{3k} < 10^{100} \iff 3k < 100 \iff k < \frac{100}{3} \iff k < 34$$

Si on ajoute 34 aux 6 itérations nécessaires pour calculer $\sqrt{2}$, on trouve 40 itérations.

4) $\ln(1+x)$ n'est calculable que pour $|x| < 1$. Il faut fonc transformer $\ln 100$ pour le calculer à partir d'une racine carrée et de $\ln(1+x)$ pour un x tel que $x < 1$ (dans le cas contraire, la suite ne converge pas) :

$$\begin{aligned}
\ln 100 &= 2\ln 10 = 2\ln 2 + 2\ln 5 = 2\ln 2 + 2\ln\left(4\frac{5}{4}\right) \\
&= 2\ln 2 + 2\ln 4 + 2\ln\frac{5}{4} = 2\ln 2 + 4\ln 2 + 2\ln\left(1+\frac{5}{4}-1\right) \\
&= 6\ln 2 + 2\ln\left(1+\frac{1}{4}\right)
\end{aligned}$$

C'est une décomposition, ce n'est pas la seule possible.

fin exo 0.0.4 □

Index

Table des matières