DS d'informatique : corrigé

PTSI Lycée Eiffel

14 novembre 2014

Exercice 1

1. On peut citer entre autres le microprocesseur (ou, si on veut être précis, le support ou socket sur lequel le microprocesseur est implanté), les mémoires vive (RAM) et morte (ROM), le chipset, le BIOS, l'horloge interne etc.
2. On peut effectuer des divisions euclidiennes comme dans le cours, ou simplement écrire $167 = 128 + 32 + 4 + 2 + 1 = 2^7 + 2^5 + 2^2 + 2^1 + 2^0$, soit 10100111 en binaire.
3. Je vous laisse reprendre le cours : libre, de haut niveau, à typage dynamique fort, multi-paradigmes et multi plate-forme, etc.

Exercice 2

1. On peut par exemple écrire ceci :

```
def somme(n) :
    s=0
    for i in range(1,n+1) :
        s=s+1/i
    return s
```

2. Complétons :

```
def f(a) :
    s=0
    i=0
    while s<a :
        i=i+1
        s=s+1/i
    return i
```

3. La variable i est de type int (nombre entier) et représente l'indice k dans la somme qu'on cherche à calculer (et donc le nombre de termes de la somme) ; la variable s sera de type float (int initialement) et représente tout simplement la somme qu'on caclule ; la variable, ou plutôt le paramètre a sera un float ou un int selon ce que l'utilisateur donnera comme valeur.
   Si on cherche à calculer $f(2)$, a prend la valeur 2 (qui ne bougera pas), i et s sont initialisés à 0. Comme s<a ($0 < 2$) on fait un premier passage dans la couble while qui affecte à i la valeur 1 et à s la valeur 1. On a toujours s<a, on effectue un deuxième passage dans la boucle, qui affecte à i la valeur 2 et à s la valeur $1 + \frac{1}{2} = \frac{3}{2}$ (ou plutôt 1.5 en Python). La condition s<a est toujours vraie, troisième passage dans la boucle pour affecter à i la valeur 3 et à s la valeur $\frac{3}{2} + \frac{1}{3} = \frac{11}{6}$ (soit environ 1.8333). On fait un quatrième passage dans la boucle, pour affecter à i la valeur 4 et à s la valeur $\frac{11}{6} + \frac{1}{4} = \frac{25}{12}$ (environ 2.0833). Ca y est, la condition s<a n'est plus vérifiée, on sort de la boucle et on retourne la valeur de i, c'est-à-dire 4.

4. La raison la plus simple est que le nombre de calculs risque d'être extrêmement élevé lorsque a est grand, et la capacité de calcul de Python n'est pas illimitée ; pour un trop grand nombre de passages dans une même boucle, il va arrêter automatiquement le calcul. Autre possibilité : au bout d'un certain temps, la valeur $\frac{1}{i}$ qu'on ajoute à la somme devient tellement faible par rapport à la valeur de s que Python l'arrondit à 0. Autrement dit, si on calcule très longtemps, Python considère que la suite (S_n) finit par être constante !

Exercice 3

1. Tout simplement (on suppose le module math importé par défaut) :

```
def f(x) :
    return x+log(x)
```

2. On ne peut pas donner de valeur décimale au pas dans un range, mais on peut toujours ruser :

```
for i in range(1,11) :
    print(f(i/10))
```

3. Cette fois-ci, il vaut mieux utiliser une boucle while, en arrêtant les calculs dès qu'on tombe sur une valeur de $f(x)$ positive :

```
x=0.01
while f(x) <=0 :
    x=x+0.01
print(x)
```

   On affichera ici une valeur approchée par défaut de α puisque la valeur affichée est la dernière pour laquelle $f(x) \leqslant 0$.

4. (a) On ne rentre même pas dans la boucle while puisque la condition n'est pas vérifiée, et le programme affiche simplement la valeur de a, soit 0.5.

   (b) Cette fois-ci, on rentre dans la boucle. On affecte à c la valeur $(a+b)/2$, soit 0.75, on calcule $f(0.75) > 0$ et on affecte à la variable b la valeur 0.75 (a ne change pas de valeur). On est prêts pour un nouveau passage dans la boucle (ou pas, ça dépend de la valeur de e).

   (c) Ce programme calcule une valeur approchée de α à une précision de e près, en effectuant une recherche dichotomique. À chaque étape, on remplace au choix a ou b par la moyenne entre les deux valeurs, en s'arrangeant pour que la fonction s'annule toujours entre a et b (on vérifie facilement que $f(a)$ est toujours négatif, et $f(b)$ toujours positif). On a donc toujours $\alpha \in [a, b]$, avec la largeur de l'intervalle $[a, b]$ qui est divisée par 2 à chaque étape, jusqu'à devenir plus petite que e. Une fois la boucle terminée, on connait donc un intervalle de largeur plus petite que e contenant la valeur α, ce qui signifie que a (la valeur affichée) est une valeur approchée de α à e près par défaut.

   (d) Puisque la largeur initiale de l'intervalle vaut $\frac{1}{2}$ et qu'elle est divisée par 2 à chaque passage, elle sera égale à $\frac{1}{2 \times 2^n}$ après n passages. Pour avoir $\frac{1}{2 \times 2^n} < \frac{1}{1000}$, il faut $2^n \geqslant 500$, ce qui est le cas pour $n = 9$. On effectuera donc neuf passages. Par la première méthode, on a besoin d'au plus 500 passages dans la boucle pour obtenir la valeur approchée (en progressant de 0.001 à chaque fois, on a au maximum 500 valeurs à tester entre 0.5 et 1). En moyenne, on en fera plutôt 250, mais ça reste largement moins bon que l'algorithme dichotomique !