Concours Blanc d'Informatique : corrigé

PTSI A et B Lycée Eiffel

7 juin 2017

Première partie : Construction d'un tableau.

1. Les entiers naturels (positifs) sont simplement stockés en binaire, en général avec une place mémoire fixée à l'avance qui limite donc naturellement le nombre d'entiers différents qu'on peut stocker (par exemple, sur 8 bits, on stockera les entiers inférieurs ou égaux à $2^8 - 1$, soit 255). Quand on veut utiliser également des entiers négatifs, on garde un stockage binaire, mais on utilise le complément à 2 pour stocker les entiers négatifs : les entiets compris entre 0 et $2^{n-1} - 1$ sont stockés normalement (et ont un premier bit nul), un entier négatif k compris entre -2^{n-1} et -1 est représenté par le code binaire de l'entier naturel $2^n + k$ (et aura donc un code dont le premier bit est un 1). Ici, le classement étant a priori inférieur à 2 000, donc en gros à 2^{11}, il faudrait environ 11 bits (probablement 12 pour ne pas prendre de risque) pour stocker le classement.
2. L'ASCII (American Standard Code for Information Interchange) est une liste de codes correspondant aux caractères les plus utilisés pour taper du texte. Initialement, l'ASCII stockait sur 7 bits les 128 caractères les plus utilisés aux États-Unis (lettres non accentuées, signes de ponctuation).
3. Un caractère va donc prendre un octet. On peut considérer qu'il y a environ 30 caractères par ligne, et on nous a déjà indiqué qu'il y avait 2 000 lignes dans le fichier, donc de l'ordre de 60 kilo-octets pour le stockage du fichier complet.
4. La commande est **f=open('joueurs.txt','z')**, le deuxième argument noté z devant en pratique être remplacé par une des trois possibilités suivantes :
   - 'r' (mode lecture) si on veut simplement récupérer des informations dans un fichier déjà existant.
   - 'w' (mode écriture) si on veut stocker des données dans un fichier initialement vierge.
   - 'a' (mode ajout) si on veut stocker des données à la suite d'un fichier déjà existant.
5. On peut utiliser brutalement la commande **readlines** et renvoyer la longueur de la liste créée, mais ça présente l'inconvénient de devoir créer une liste qui va ici être très grosse. Pour faire plus léger, on peut récupérer les lignes l'une après l'autre :

```
f=open('joueurs.txt','r')
n=0
for i in f.readline() :
        n=n+1
print(n)
f.close()
```

6. On va écrire un programme proche du précédent, en traitant chaque ligne à la suite :

```
f=open('joueurs.txt','r')
liste=[]
for i in f.readline() :
```

```
        c=i.split()
        liste.append(c[0])
print(liste)
f.close()
```

7. On remplace la ligne liste=[] par tetesdeserie,anonymes=[],[] et on ajoute les lignes suivantes juste après la ligne c=i.split() du programme précédent :

```
n=c[2]
if n<33 :
    tetesdeserie.append(c[0])
elif n<129 :
    anonymes.append(c[0])
```

   Et bien entendu, on affiche les deux tableaux en fin de programme.

8. Beaucoup de possibilités, par exemple :

```
tableau=range(1,129)
```

9. On peut importer directement tout le module via la commande import machin (machin étant le nom du mdule, on peut aussi le renommer en ajoutant as bidule derrière la commande d'importation), ce qui force à rappeler le nom du module à chaque fois qu'on utilise une commande qui en est issue (par exemple, si on a importé ainsi le module random et qu'on souhaite utiliser sa fonction randint, on devra taper random.randint(a,b)). Sinon, on peut importer une fonction (sans avoir besoin du nom de module à l'aide de la commande from machin import trucmuche (trucmuche étant bien sûr ici le nom de la fonction), ou même from machin import * si on veut salement importer toutes les fonctions du module d'un seul coup.

10. Une façon de faire est de commencer par les têtes de séries, en tirant à chaque fois un joueur aléatoirement dans la liste correspondante et surtout en le supprimant de la liste ensuite. On remplit ensuite le reste à l'aide des anonymes. Le programme suivant est particulièrement hideux, mais ça marche.

```
n=31
for i in range(16) :
    j=tetesdeserie.randint(0,n)
    tableau[8*i]=j
    tetesdeserie.remove(j)
    n=n-1
    j=tetesdeserie.randint(0,n)
    tableau[8*i+7]=j
    tetesdeserie.remove(j)
    n=n-1
n=96
for i in range(127) :
    if tableau[i]==i :
    j=anonymes.randint(0,n)
    tableau[i]=j
    anonymes.remove(j)
    n=n-1
```

Deuxième partie : Étude d'une équation différentielle.

1. On intègre brillamment à partir de la condition $x'' = 0$, et en exploitant les conditions initiales : $x'(t) = 30$, puis $x(t) = 30t$.
2. La méthode d'Euler du **premier** ordre consiste à calculer de proche en proche des valeurs approchées des valeurs prises par la solution d'une équation différentielle, en approchant cette solution sur le segment séparant deux valeurs successives par sa tangente (elle-même approchée), dont on calcule la pente approchée à partir de la dernière valeur calculée et de l'équation différentielle. Ce principe s'adapte sans difficulté à un système de deux équations différentielles du premier ordre à deux inconnues, et on peut ensuite transformer une équation différentielle du **deuxième** ordre en système de deux équations du premier ordre vérifiée par la fonction inconnue y et par sa dérivée y'.
3. Il est complètement idiot d'utiliser la méthode d'Euler pour cette équation, mais peu importe, c'est ce qu'on nous demande ! Bien entendu, puisque toutes les valeurs des constantes sont connues, pas besoin de faire intervenir plein de variables comme dans le programme vu en cours. En posant $y(t) = z'(t)$, l'équation peut en tout cas se ramener au système constitué des deux équations du premier ordre $y = z'$ et $y' = -9.81$. Voici un programme effectuant ce qui est demandé :

```
import matplotlib.pyplot as plt
def Eulertennis() :
    z,y,t=1,10,0
    fonction, derivee, temps = [z],[y],[t]
    for i in range(300) :
        z,y=z+0.01*y,y=y-9.81*0.01
        t=t+0.01
        temps.append(t)
        fonction.append(z)
        derivee.append(y)
    plt.plot(temps,fonction)
    plt.show()
```

   On devrait normalement observer une trajectoire parabolique.
4. Manifestement, on a voulu prendre une certaine marge, probablement pour prendre en compte le fait que l'altitude maximale de la balle n'est pas toujours atteinte juste au-dessus du filet, mais les critères choisis sont complètement arbitraires et grotesque. Bien sûr, la balle ne franchira jamais le filet si le maximum d'altitude est inférieure à la hauteur du filet, mais la réciproque est grossièrement fausse car on ne tient pas du tout en compte la trajectoire horizontale de la balle. Si on tape très fort mais verticalement au-dessus de soi, on aura un maximum très élevé mais la balle n'atteindra jamais le filet par exemple.

   Pour le programme demandé, il faut d'abord calculer le maximum de la liste avant d'afficher le message (on ne veut pas un message pour chaque valeur de la liste) :

```
def filet(altitudes) :
    m=0
    for i in altitudes :
        if i>m :
            i=m
    if m >=1.2 :
        return('La balle a franchi le filet')
```

```
        elif m<=1.1 :
            return('La balle a lamentablement échoué dans le filet')
        return('On ne peut pas savoir, les données sont trop imprécises')
```

5. Toutes les méthodes permettant une résolution approchée d'équation peuvent convenir : dichotomie, méthode de Newton, méthode des sécantes etc.
6. On va donc chercher dans la liste **altitudes** la première valeur négative, en faisant une dichotomie sur les indices :

```
a,b=0,len(altitudes)-1
while b-a>1 :
    c=(a+b)//2
    if altitudes[c]>0 :
        a=c
    else :
        b=c
    return 30*0.01*c
```

   La multiplication finale sert à transformer l'indice en temps écoulé (multiplication par 0.01, puisque le pas est supposé de 0.01 seconde), puis à calculer la valeur de x correspondante (on a vu plus haut que $x(t) = 30t$).
7. Comme me l'a fort justement fait remarquer mon excellent collègue M.Raimi, j'ai complètement craqué sur cette question, puisque je prétends que le frottement devrait être proportionnel à la vitesse (ce qui est bien ce que je voulais !), mais que j'ai mis des équations où on a une proportionnalité avec les valeurs des fonctions, ce qui est évidemment physiquement ridicule (le frottement est proportionnel à la hauteur atteinte par la balle, c'est pas mal ça non ?). Bref, ça ne change rien à la question. Tout ce qu'on vous demandait d'expliquer, c'est qu'on avait maintenant un système de deux équations différentielles du deuxième ordre. Pour le traiter par la méthode d'Euler, on le transforme « simplement » en système de quatre équations différentielles du premier ordre : on pose $w(t) = x'(t)$ et $y(t) = z'(t)$, et on ajoute à ces deux équations les deux équations suivantes : $w'(t) = -k\sqrt{w(t)^2 + y(t)^2}w(t)$, et $y'(t) = -9.81 - k\sqrt{w(t)^2 + y(t)^2}y(t)$ (j'ai remis des vraies équations de frottements, si on garde la proportionnalité aux fonctions comme dans l'énoncé, on remplace juste w et y à la fin des équations par x et z).

Troisième partie : Exploitation d'une base de données.

1. CREATE TABLE Tournois (Idtournoi INTEGER PRIMARY KEY, Ville VARCHAR(30), Pays VARCHAR(30), Datededébut DATE, Prix INTEGER)
2. ALTER TABLE Joueurs ADD main BOOLEAN (on considèrera que 1 corrspond à droitier et 0 à gauche par exemple).
3. Non, car la relation entre les deux entités correspondantes est très probablement de cardinalité trop compliquée : un même joueur peut évidemment gagner plusieurs tournois, mais un tournoi peut également avoir plusieurs vainqueurs (le même tournoi ayant une édition chaque année). Les attributs Vainqueur et Finaliste sont des clés secondaires de la table Vainqueurs, référeçant tous les deux l'attribut Idjoueur de la table Joueurs (c'est le numéro d'identification qu'on indiquera et pas le nom du joueur, qui ne constitue pas une clé prmaire).
4. SELECT Ville, Datededébut FROM Tournois WHERE Pays='France'
5. SELECT COUNT(Ville) FROM Joueurs, Tournois, Vainqueurs WHERE Vainqueurs.Idtournoi=Tournois.Idtournoi AND Vainqueur=Idjoueur AND Nom='Djokovic' AND Prénom='Novak' AND Année=2015

6. SELECT Nom,Prénom FROM Joueurs, Tournois, Vainqueurs WHERE Vainqueurs.Idtournoi=Tournois.Idtournoi AND Vainqueur=Idjoueur AND Prix>1000000 (cette solution n'est pas optimale car un joueur ayant gagné plusieurs tournois dotés de plus d'un million de dollars apparaitra plusieurs fois dans la liste, il existe des options permettant d'éviter ceci mais on n'en a pas parlé en cours).