Concours Blanc : corrigé

PTSI A et B Lycée Eiffel

28 mai 2014

Première partie : Bases de données.

1. Pour la table garages, l'attribut n°id est manifestement fait pour servir de clé primaire. On peut imaginer que l'atribut adresse puisse aussi servir de clé primaire si on est tordu. Pour la table voitures, rien d'évident, mais seul l'attribut nom du modèle peut raisonnablement être unique pour chaque voiture, et donc servir de clé primaire. Enfin, la table ventes est issue d'une association entre les entités garages et voitures, sa clé primaire est donc le coupe (n°id, nom du modèle).
2. Dans l'algèbre relationnelle : $\Pi_{nomdumodele}(\sigma_{nombredeplaces \geqslant 5}(voitures))$. En SQL : SELECT nomdumodele FROM voitures WHERE nombreplaces $>$ 4.
3. SELECT AVG(prix) from vente,garages WHERE vente.n°id=garages.n°id AND garages.n°id=3
4. Dans l'algèbre relationnelle :
   $\Pi_{marque}(\sigma_{ville=Bordeaux}((garages \bowtie_{garages.n°id=vente.n°id} vente) \bowtie_{vente.nomdumodele=voitures.nomdumodele} voitures))$
   En SQL : SELECT marque FROM garages,voitures,vente WHERE vente.n°id=garages.n°id AND voitures.nomdumodele=vente.nomdumodele AND garages.ville='Bordeaux'
5. ALTER TABLE vente ADD anneedeconstruction : integer
6. SELECT voitures.nomdumodele,adresse FROM voitures,vente,garages
   WHERE vente.n°id=garages.n°id AND voitures.nomdumodele=vente.nomdumodele
   AND occasion=True AND anneedeconstruction $>$ 2008

Deuxième partie : Programmation en Python.

1. C'est du cours ! On dispose du mode lecture ('r' pour read en Python), qui permet d'accéder au contenu du fichier sans le modifier ; du mode écriture ('w' pour write) qui permet d'écrire dans un fichier initialement vierge ; et du mode ajout ('a' pour add) qui permet d'ajouter du contenu à la fin d'un fichier déjà existant.
2. Il s'agit de la méthode append.
3. ```
   > fichier=open(tutures.txt,'r')
   > modeles=[],prix=[]
   > for i in fichier.readlines() :
   >         l=i.split(';')
   >         modeles.append(l[0])
   >         prix.append(l[1])
   >         fichier.close()
   ```
4. ```
   > def miniliste(l) :
   > m=l[0]
   > for i in l[1 :] :
   ```

```
>          if i<m :
>                m=i
> return m
```

5. La variable m est initialisée à la valeur correspondant à l[0], c'est-à-dire 7. La variable i prend alors successivement les valeurs 5, 12, 1 et 8 : lorsque i=5, on change la valeur de m, qui est désormais aussi égale à 5 ; lorsque i=12, on ne fait rien ; lorsque i=1, on change à nouveau m qui prend la valeur 1 ; lorsque i=8 on ne fait rien. On ressort alors la valeur 1.
6.
```
> def moyenne(l) :
> s=l[0],n=1
> for i in l[1 :] :
>          s=s+i
>          n=n+1
> return s/n
```
7. Il est évidemment préférable de ne calculer qu'une seule fois la moyenne :

```
> m=moyenne(prix)
> l=[]
> for i in range(len(prix)) :
>          if prix[i]<m/2 :
>                l.append(modeles[i])
```
8.
```
> def ecarttype(l) :
>          m=moyenne(l)
>          s=0
>          for i in l :
>                s=s+(i-m)**2
>          return sqrt(s/len(l))
```
9. Pour pouvoir réaliser cela, il faut conserver non seulement la valeur minimale dans la liste, mais surtout la position de la liste où a été atteint ce minimum. On peut par exemple procéder comme ceci :

```
> m=prix[0],a=modeles[0]
> for i in range(1,len(prix)) :
>          if prix[i]<m :
>                m=i
>                a=modeles[i]
> return a
```
10. Il affichera la première voiture dans la liste correspondant à ce prix minimal. Si on veut une liste de tous les modèles, il faut remplacer la variable a par une liste, et gérer le cas d'égalité du prix avec le minimum :

```
> m=prix[0],a=[modeles[0]]
> for i in range(1,len(prix)) :
>          if prix[i]<m :
>                m=i
>                a=[modeles[i]]
>          elif prix[i]==m :
>                a.append(modeles[i])
> return a
```
11. On effectue en gros $2n$ comparaisons (deux pour chaque élément de la liste). La première version ne faisait que n comparaisons.

Troisième partie : Analyse numérique.

1. La méthode d'Euler consiste à résoudre numériquement une équation différentielle, et plus précisément à obtenir une courbe approchée de la courbe solution, qui sera une courbe de fonction affine par morceaux, chacun des morceaux étant obtenu en traçant la tangente (approchée à partir du deuxième morceau) à la courbe au point qui est la borne supérieure du segment précédent (la pente de cette tangente étant obtenue à partir de la valeur approchée de la fonction à cet endroit en expoitant l'équation différentielle). Dans le programme, la fonction f représente l'équation différentielle, qui doit être mise sous la forme $v'(t) = f(t, v(t))$. Le réel v_0 est une condition initiale pour la vitesse, et le réel dt correspond au pas utilisé entre deux valeurs successives de notre approximation.
2. > v=v+dt*f(t,v)
   > t=t+dt
   > temps.append(t)
   > vitesse.append(v)
3. On utiliserait le module matplotlib.pyplot, en l'important via la commande import matplotlib.pyplotas plt, et en traçant la courbe avec la command plt.plot(temps,vitesse).
4. > def f(t,v) :
   > hspace1cm return α - β*v**2
5. > i=0
   > while vitesse[i] >0 :
   > hspace1cm i=i+1
   > print(i)
6. Il est bien sûr beaucoup plus efficace de procéder par dichotomie :

   > i,j=0,500
   > while j-i>1 :
   > hspace1cm c=(i+j)//2
   > hspace1cm if vitesse[c]>0 :
   > hspace2cm i=c
   > hspace1cm else :
   > hspace2cm j=c

   Puisqu'on divise le nombre de valeurs par 2 à chaque étape, il nous restera 250 valeurs une étape, 125 après deux étapes etc, et on descendra à 1 après neuf étapes environ puisque $2^9 > 500$. La première méthode utilisée nécessitait environ 250 comparaisons (si le passage à des valeurs négatives se trouve au milieu de la liste), ce qui est évidemment nettement moins efficace.
7. Ah ben en fait, on l'a déjà fait à la question précédente.