DS3 : corrigé

MP Lycée Camille Jullian

11 février 2022

Première partie : base de données des animaux du zoo.

1. CREATE TABLE animaux (n°id INTEGER PRIMARY KEY, espece VARCHAR(30), nom VARCHAR(30), age INTEGER, Date DATE)
2. Dans la mesure où il est tout à fait possible qu'un même employé effectue plusieurs types de soins différents sur un même animal, on ne peut pas se contenter de prendre par exemple le couple constitué des numéros d'identifiants de l'employé et de l'animal comme clé primaire. En fait, le plus simple est de créer de toutes pièces un numéro qu'on appellera typiquement nn° soin. Les clés secondaires seront le nn° employe du soigneur et le nn° id de l'animal, qui doivent apparaitre comme attribut dans cette table.
3. SELECT nom, age FROM animaux WHERE espece='éléphant'
4. SELECT COUNT(nticket) FROM ticket WHERE tarif='plein tarif' AND date='2018/05/12'
5. SELECT Employes.nom, prenom FROM (Employés INNER JOIN Soins ON Employés.n°employe=Soins.n°employe) INNER JOIN animaux ON animaux.n°id=Soins.n°id WHERE animaux.nom='Baloo'

Deuxième partie : exploitation d'un fichier de données et programmation Python.

1. La syntaxe est f=open('fichier.txt','m'), pour créer une variable f qui pointe vers le fichier texte nommé fichier.txt. Le caractère m doit être remplacé par une des trois possibilités suivantes : 'r' pour lire le fichier sans le modifier, 'a' pour ajouter du contenu à la fin du fichier (sans modifier ce qui était déjà dans le fichier) ou 'w' pour écrire dans un fichier vierge.
2. L'énoncé racontait n'importe quoi, puisque split est une méthode et non une fonction. On va respecter la syntaxe Python dans le programme qui suit :

```
def listealpha() :
    l=[]
    f=open('animaux.txt','r')
    for i in f.readlines() :
        a=i.split(';')
        l.append(a[1])
    l.sort()
    return l
```

3. On va essayer de faire quelque chose d'un peu précis en profitant du fait qu'on a la date exacte d'arrivée de l'animal au zoo :

```
def listevieux() :
    l=[]
```

```
f=open('animaux.txt','r')
for i in f.readlines() :
    a=i.split(';')
    b=a[3]
    if b[2 :4]=='01' or (b[2 :4]=='01' and int(b[ :2])<12) :
        t=2022-int(b[-4 :])
    else :
        t=2022-int(b[-4 :])-1
    if t>int(a[2])/2 :
        l.append(a[1])
return l
```

Troisième partie : étude de la température du logement des girafes (programmation Python).

1. Les nombres flottants sont codés en binaire en séparant le codage en trois parties : un bit de signe sert à donner le signe du flottant, puis un certain nombre de bits servent à coder un entier relatif e appelé exposant en complément à 2, et les bits restants servent à coder un entier positif m nommé mantisse en binaire naturel. Le code obtenu sert alors à coder le nombre $x = \pm m \times 2^e$. Autrement dit, la mantisse représente tous les chiffres significatifs (en binaire) du nombre x, et l'exposant « déplace la virgule »).
2. Une capacité de 2 octets correspond à 16 bits, soit 2^{16} valeurs possibles. On sait que $2^{10} \simeq 1000$, donc $2^{16} \simeq 64\ 000$, ce qui laisse espérer quatre chiffres significatifs corrects en écriture décimale (à condition d'optimiser le stockage, en pratique on n'aurait probablement pas plus de trois chiffres significatifs).
3. Si on veut deux décimales dans une amplitude maximale de 10 degrés, on aura au maximum $10 \times 100 = 1\ 000$ valeurs differentes à stocker, ce qui doit pouvoir se faire sur un octet en optimisant le stockage. On aura donc besoin de 12 octet par heure, 288 octets par jour, et un peu plus de 2 ko pour une semaine. Là encore, en pratique, il faudrait sûrement multiplier cette valeur par 10 pour être réaliste.
4.
```
import matplotlib.pyplot as plt
plt.plot(horaires, temperatures)
plt.show()
```
5.
```
def moyenne(L) :
    s=0
    for i in L :
        s=s+i
    return s/len(L)
```

   Si on ne veut pas utiliser len, on créer une deuxième variable en même temps que s, initialisée à 0 et qu'on augmente d'une unté à chaque passage dans la boucle, pour retrouver le nombre total d'éléments de la liste. Dans tous les cas, on a une boucle contenant des opérations à coût constant, donc une complexité linéaire en $O(n)$, où n représente bien sûr le nombre d'éléments de la liste.

6.
```
def variance(L) :
    m=moyenne(L)
    a=0
    for i in L :
```

```
        a=a+(i-m)**2
    return a/len(L)
```

Le programme effectue une seule fois le calcul de la moyenne, qui est linéaire, et le passage dans la boucle nécessite à nouveau une quantité linéaires de calculs. On reste donc avec une complexité linéaire en $O(n)$.

7.
```
def checktemp(L) :
    m=moyenne(L)
    if m<25 or m>28 :
        return False
    if variance(L)>3 :
        return False
    return True
```

8. (a) Notre élève est manifestement une vraie quiche en info, il a commis beaucoup d'erreurs en quelques lignes :
   - à la troisième ligne, la syntaxe est incorrecte, il faudrait un « for i in range(len(L)) »
   - la variable e définie à la quatrième ligne est censée être égale à l'écart-type et pas à la variance pour que la formule soit ensuite correcte
   - la ligne 5 comporte deux erreurs, la variable s est réinitialisée au lieu d'être modifiée (soit on remplace le = par un +=, soit on ajoute un +s en fin de ligne), et la formule est mal recopiée puisqu'on devrait avoir un cube à la place du carré au numérateur
   - le calcul effectué dans le return est incorrect, il faut effectuer s*n/(n-1)/(n-2) ou s*n/((n-1)*(n-2)). De plus, la variable n, manifestement égale à len(L), n'a pas été définie.

   (b) Il n'y a qu'une seule boucle, mais la moyenne et la variance sont recalculées à chaque fois, donc le temps de calcul à l'intérieur de la boucle est linéaire, et le programme lui-même quadratique, complexité en $O(n^2)$.

   (c) Il suffit de sortir de la boucle les calculs de moyenne et de variance pour rendre le programme linéaire.

9.
```
def detectionanomalies(h,t) :
    n=0
    l=[]
    for i in range(len(t)) :
        if t[i]>=22 :
            n=0
        else :
            n=n+1
            if n==3 :
                l.append(h[i])
    return l
```

Quatrième partie : étude de la courbe de croissance d'un bébé panda (analyse numérique).

1. Si $\beta = 0$, on a $P'(t) = \alpha P(t)$, équation différentielle dont les solutions sont des fonctions exponentielles. Si on veut que le poids du bébé soit croissant, on va donc imposer $\alpha > 0$, mais il faudra prendre $\beta < 0$ pour éviter que le poids ne grandisse trop violemment.

2. On découpe l'intervalle de résolution en n intervalles de même largeur, et on approche sur chacun de ces intervalles la courbe recherchée par une tangente approchée dont la pente est obtenue à partir de la valeur approchée calculée au point précédent, en exploitant bien sûr l'équation différentielle qui relie la fonction et sa dérivée.

3.
```
import matplotlib.pyplot as plt
def simulpanda(a,b) :
    temps,poids=[0],[0.1]
    for i in range(1000) :
        poids.append(poids[-1]+0.01*(a*poids[-1]+b*poids[-1]**2))
        temps.append(temps[-1]+0.01)
    plt.plot(temps,poids)
    return poids
```

4. On a utilisé dans le programme précédent un découpage qui renvoie 1 000 valeurs du poids pour les 10 premiers jours, il faut donc extraire les 10 valeurs qu'on veut comparer avec le vrai poids mesuré :

```
def valide(a,b,poids) :
    theorie=simulpanda(a,b)
    for i in range(1,11) :
        if abs(poids[i]-theorie[100*i-1])/poids[i]>0.05 :
            return False
    return True
```

5. (a)
```
def approxder(poids,delta)
    l=[]
    for i in range(1,len(poids)-1) :
        l.append((poids[i-1]+poids[i+1])/(2*delta))
    return l
```

(b) Il faut enlever les valeurs extrêmes de la liste poids pour avoir le même nombre d'éléments que dans l'autre liste, puis faire une approximation polynomiale de degré 2. Cette approximation va nous ressortir trois coefficients, on se contentera de prendre les deux derniers pour nos valeurs de α et β (si le modèle n'est pas pourri, le coefficient constant devrait de toute façon être proche de 0).

```
P=polyfit(approxder(poids,delta),poids[1 : :-1],2)
(a,b)=(P[1],P[2])
```

Le principal défaut est bien sûr qu'on ne peut pas imposer un coefficient constant nul dans polyfit, ce qui fait que si on obtient une valeur non négligeable, on ne pourra pas déduire grand chose du résultat obtenu (sauf bien sûr que le modèle n'est peut-être pas le meilleur possible).