#Correction du CNC 2015

Exercice 1 : base de données

#le schéma de la base est le suivant :

Train(immatriculation,gare_attache,id)

Trajet(num_trajet,immatriculation,ville_dep,ville_arr,heure_dep,heure_arr)

type(id,nom,nb_place)

#**************************************

question 1 :

#

#

#requete : select heure_dep,heure_arr,immatriculation from Trajet where ville_dep='fez';

#**********************************

#Question 2:

#

#

#requete : select distinct heure_dep,heure_arr,trajet.immatriculation,gare_attache from Trajet join train on Trajet.immtriculation=Train.immatriculation where gare_dep='fez';

#Question 3:

#

#

#requete : select id, count(num) from (select Trajet.immatriculation as num,id from Trajet join train on Trajet.immtriculation=Train.immatriculation where ville_dep='Rabat') group by id;

#**********************************

#Question 4:

#

#

```
#requete :  select  nom,gare_dep from (Trajet join train on
Trajet.immtriculation=Train.immatriculation  where heure_dep between 7:00 and 13:00)join type on
Train.id=type.id;

#*********************************

#

#Question 5

import sqlite3

cnx=sqlite3.connect("gestion_trains.sqlite")

cur=cnx.cursor()

requete="select heure_dep,heure_arr,immatriculation from Trajet where ville_dep='fez';"  # requete
question 1

#cur.execute(requete)

#resultat=cur.fetchall()

#for train in resultat:

 #   print("numero : ",train[2]," heure depart",train[0]," heure _arr :",train[1])

cnx.close()


#***************************

# en posant Y=[y,y1] avec y1 derivé premier de y par rapport à t

# dont Y1 derivé de Y sera le vecteur [y1,y2] avec y2 derivé seconde de yet y1' derivé premiere de y

# prenant Y1=[y1,y2]=[[0,1],[a,b]]*[y,y1]=[[0,1],[a,b]]*Y=F(Y,t) qui est une equation differentielle de
degré 1

#****************************#

#

#  Question 7

#

import numpy as np

from scipy.integrate import odeint

import matplotlib.pyplot as plt
```

```
def F(z,t):   # z est le vecteur Y definit dans la question précédente et t la variable muette.
    return np.array([z[1],-3*z[0]-z[1]])


#************************************
#
#Question 8
#
Lx=np.linspace(0,5,100)
Y0=[0.05,0.7]
LL=odeint(F,Y0,Lx)
Ly=LL[:,0]
plt.title("figure2")
plt.plot(Lx,Ly)
plt.show()


#*******************************************
# probleme 1
# Question 9
#
# un interet majeur lors de l'utilisation de la notation postfixé est celui de ne pas utiliser de
parenthèses
# et de ne pas tenir compte de laa priorité des opérateurs.
#
#**********************************
# Question 10
#
def initPile():
    return []
```

```
#
#**********************************
# Question 11
#
def estVide(pile):
    return pile==[]
#
#**********************************
# Question 12
#
def empiler(pile,elem):
    pile.insert(0,elem)
    return pile
#
#**********************************
# Question 13
#
def depiler(pile):
    if not estVide(pile):
        x=pile[0]
        pile.remove(pile[0])
        return x
    else:
        raise pileVideException
#
#**********************************
# Question 14
```

```
#
def valeurSommet(pile):
    if not estVide(pile):
        return pile[0]
    else:
        return None
#
#**********************************
# Question 15
#
def hauteur(pile):
    return len(pile)
#
#**********************************
# Question 16
#
def estEntier(n):
    n=str(n)
    if len(n)==1:
        if n>'9' or n<'0':
            return False
        else:
            return True
    else :
        return estEntier(n[0]) and estEntier(n[1:])


#
```

```
#**********************************
# Question 17
# variante verifiant si un nombre es entier ou non
def estNombre(n):
    a=type(n)
    print(a)
    return 'int' in str(a)
#**********************************
# Question 17
#
def eval(operateur,operande1,operande2):
    if estNombre(operande1) and estNombre(operande2):
    #if estEntier(operande1) and estEntier(operande2):
        if operateur in "+-*/.":
            if operateur=="+":
                return operande1+operande2
            elif operateur=="-":
                return operande1-operande2
            elif operateur=="*":
                return operande1*operande2
            elif operateur=="/":
                return operande1/operande2
            elif operateur==".":
                return
        else:
            raise OpNonValideException
    else:
```

```
        raise ArgNonValideException




#
#**********************************
# Question 18
#
def evalue(L):
    pile=initPile()
    for x in L:
        if x!='.':
            if estEntier(x):
                pile=empliler(pile,x)
            elif x in "+-*/":
                a=depiler(pile)
                b=depiler(pile)
                res=eval(x,a,b)
                pile=empiler(pile,res)
        else:
            res=depiler(pile)
            return res


#
#**********************************
# Question 19
#
```

```
def estBienParenthesee(L):
    parouv=0
    parfer=0
    for x in L:
        if x=='(':
            parouv+=1
        elif x==')':
            parfer+=1
    if parouv==parfer:
        return "Expression bien parenthésée"
    else:
        return "Expression mal parentésée"
#on pourra aussi faire
    # if L.count("(")==L.count(")"):
    #     return "Expression bien parenthésée"
    # else:
    #     return "Expression mal parentésée"
#
#**********************************
# Question 20
#
def transInfix(L):
    # L.reverse()
    # L.pop(0)
    # L.append('.')
    pile=initPile()
    OpOrdre=['+','-','*','/']
```

```
    Lp=[]
    for x in L:

        if x in "+-*/":
            if estVide(pile):
                empiler(pile,x)
            else:
                oppile=valeurSommet(pile)
                posx=OpOrdre.index(x)
                pospile=OpOrdre.index(oppile)
                while posx<=pospile and not estVide(pile):
                    v=depiler(pile)
                    Lp+=[v]

                empiler(pile,x)

        else:
            if x!='.':
                Lp= Lp+ [x]

    while not estVide(pile):
        v=depiler(pile)
        Lp.append(v)
    Lp.append('.')
    print(x,pile,Lp)
    return Lp
#******************
```

#Question

#

on parcours la liste contenant l'expression infixe

si l'élément est une opérande on l'ajoute à la liste poste fixe qui initialement vide

si l'element est un opérateur (+,-,*,/) et la pile est vide on le met dans la pile

si l'element est une parenthèse ouvrante on l'empile aussi

si la pile n'est pas vide et l'element est un operateur on regarde la priorité de l'operateur au sommet de la pile à celui de l'element si l'element est plus prioritaire on l'empile sinon on dépile l'element au sommet de la pile et on l'ajoute à la liste postfixe puis on empile l'element

si l'lement est une parenthèse fermante on depile et on met dans la liste postfixe a tours de role tout les opérateurs jusqu'a atteinte de la parenthèse ouvrante corespondante qu'on depile aussi

et on continu ainsi jusqu'a la fin de la liste infixe

#

#

#******************

#Question 22

#

```
def transTotal1(Lf1):
    Lf2=[]
    #K=transInfix(Lf1)
    Ops=[]
    for x in Lf1:
        if x in '+-*/.':
            Ops.append(x)
        else:
            Lf2.append(x)
    Lf2.reverse()
    Lf2.extend(Ops)
```

```
    return Lf2



#******************
#Question 23
#
def transTotal(L):
    K=transInfix(L)
    K1=transTotal1(K)
    return K1
#*********************************
# Programme principal
#
#L1=['3','*','5','+','6','/','2','.']
L1=['1','+','3','*','4','+','8','.']
#M=transInfix(L1)
M=['8','4','+','3','*','1','-','.']
print(transTotal(L1))
```