```
##TP8
## partie 1 travail avec les Bdd

#Q1
demande_liste_activites ="SELECT activites FROM INDUSTRIE WHERE Type = 'granulaire' "
nom_activite='Extraction charbon'
demande_parametres = "SELECT stock , Trans, Ecoul FROM INDUSTRIE2 WHERE activites =
"+ str(nom_activite) + " "
#cette syntaxe permet de faire appel à une variable (‘+ variable +’)

## avec python
import os # Import des commandes permettant de manipuler la base de données
os.chdir('C:/Users/fabinou/Desktop')

import sqlite3 # Import des commandes permettant de manipuler la base de données
basesql = u"base_industries.s3db" # Base de données initiale
cnx = sqlite3.connect(basesql )
curseur = cnx.cursor ()
requete = "SELECT * FROM INDUSTRIE"
curseur.execute(requete)
data = curseur.fetchall()
print (data)

#Q2
#la clé primaire

#Q3
def interroge_bdd(requete) :
    basesql = u"base_industries.s3db" # Base de données initiale
    cnx = sqlite3.connect(basesql )
    curseur = cnx.cursor ()
    curseur.execute(requete)
    tab = curseur.fetchall()
    return tab

resultat1=interroge_bdd(demande_liste_activites)
resultat2=interroge_bdd(demande_parametres)
# import numpy as np
resultat1=np.array([['charbon','granulaire','tas','camions','laminaire'],
['beton','granulaire','sac','camions','fluide'],
['paracétamol','granulaire','sachets','trains','fluide']])
#

def choix(tableau):
    num_mat=int(input("quel numéro de matériau ? "))
    stcokage,transport,
ecoulement=tableau[num_mat,2],tableau[num_mat,3],tableau[num_mat,4]
    return [stcokage,transport, ecoulement]

print(choix(resultat1))

#Q4 Jointure
demande_prix="SELECT p.prix_tonnes FROM PRIX as p JOIN INDUSYTRIE  as i ON
p.id_activte=i.id_activite WHERE i.Type = 'granulaire' GROUP BY i.activites ORDER BY
p.prix_tonnes "

## TP8 - partie 2
##forme du tas
from random import random #pour la generation d'un nombre aleatoire

def calcul_n(h):
    n=int((h+2.0)/2*random())+1
    return n

print(calcul_n(10))
##
def initialisation(P):
```

```
    piles=[]
    for i in range(P):
        piles.append(0) #on ajoute P termes nuls à la liste des piles
    return piles

print(initialisation(20))

##
def actualise(piles,perdus):
    P=len(piles)
    i=0
    #parcours des piles 0 à P-2:
    while i<P-1:
        h=piles[i]-piles[i+1]
        if h>1: #Règle : le grain ne tombe pas si h=1 ou 0
            n=calcul_n(h)
            piles[i]=piles[i]-n #n grains tombent de la pile i à droite
            piles[i+1]=piles[i+1]+n #n grains arrivent sur la pile droite i+1
        i=i+1
    #parcours de la derniere pile P-1:
    h=piles[P-1] #hauteur de la derniere pile
    if h>1: #Règle : le grain ne tombe pas si h=1 ou 0
        n=calcul_n(h) #calcul de n pour cette derniere pile
        piles[P-1]=piles[P-1]-n #n grains se perdent
        perdus=perdus+n #MAJ cumul des grains perdus
        #debug:print( " h=", h, "piles=",piles, " cumul grains perdus=", perdus)
        return [piles, perdus] #on retourne une liste mise à jour
    else:
        return [piles, perdus]

piles=[50,0,0,0,0]
print(actualise(piles,0))
#[[34, 12, 2, 0, 1], 1]

##
def calcul_piles(P, seuil_perte,nb_actu):
    #creation d'un support vide 'piles' à P piles
    piles=initialisation(P)
    #pas de grain perdu pour démarrer
    perdus=0
    res=[piles, perdus] #liste constituée de la liste 'piles' et de l'entier 'perdus'

    while perdus<seuil_perte:
        for i in range(nb_actu):
            res=actualise(piles, perdus) #on actualise piles et perdus 10 fois
        #on ajoute alors 1 grain sur la premiere pile
        piles=res[0]
        piles[0]=piles[0]+1
        #on regarde le nbre de grains perdus
        perdus=res[1]
    #seuils grains au moins sont sortis du support à ce stade
    return piles

print(calcul_piles(20,1000,10))

##tracé simple avec matplotlib :
import matplotlib.pyplot as plt # ça peut servir ...
P0=15
seuil_perte0=1000
nb_actu0=10

X=np.arange(0,P0) #construire un vecteur [0,1,...,P-1]
Y=np.array(calcul_piles(P0, seuil_perte0,nb_actu0)) #vecteur constitué de [piles[0],
[piles[1],...,piles[P-1]]
#mettre des titres aux axes et au graphique
plt.xlabel ('Piles du support')
plt.ylabel ('Hauteurs')
```

```
plt.title ('Allure du demi-tas en fin de simulation')
#legende et correspondances des couleurs
plt.plot (X, Y,'b*',label=u'Y(X)') #graph, pour rafraichir
plt.legend(loc=1) #emplacements loc: 1,2,3,4=hautD,hautG,basG,basD respectivement

##pour revoir des GUI (pas demandé au TP)
##décommenter toute la partie ci dessous et commenter le tracé précedent
# from matplotlib.widgets import Slider #, Button, RadioButtons inutilisé ici
# plt.close('all') #ferme les graphes précédents
# #ajustement position graphe et plages des axes
# plt.subplots_adjust(left=0.15, bottom=0.25)
# plt.axis([0, 50, 0, 50])
# graph, =plt.plot (X, Y,'b*',label=u'Y(X)') #graph, pour rafraichir
# plt.legend(loc=1) #emplacements loc: 1,2,3,4=hautD,hautG,basG,basD respectivement
# #tracé du contour des piles :
# Xcontours=np.arange(2*P0) #vecteur [0,1,...,2P-1]
# Ycontours=np.arange(2*P0) #vecteur [0,1,...,2P-1]
# for i in range(0, P0):
#     Xcontours[2*i]=i
#     Xcontours[2*i+1]=i+1
#     Ycontours[2*i]=Y[i]
#     Ycontours[2*i+1]=Y[i]
# graph_contours, =plt.plot (Xcontours, Ycontours,'b-',label=u'Ycontours(Xcontours)')
#
# P=P0
# seuil_perte=seuil_perte0
# #fonction associée au slider P:
# def update_P(val):
#     global P #on va modifier P en dehors de la fonction
#     P=int(s_P.val) #lecture P sur le slider
#     X=np.arange(0,P)
#     Y=np.array(calcul_piles(P, seuil_perte0,nb_actu0))
#     graph.set_xdata(X) #MAJ tracé
#     graph.set_ydata(Y) #MAJ tracé
#     Xcontours=np.arange(2*P)
#     Ycontours=np.arange(2*P)
#     for i in range(0, P):
#         Xcontours[2*i]=i
#         Xcontours[2*i+1]=i+1
#         Ycontours[2*i]=Y[i]
#         Ycontours[2*i+1]=Y[i]
#     graph_contours.set_xdata(Xcontours) #MAJ tracé
#     graph_contours.set_ydata(Ycontours) #MAJ tracé
# #slider P:taille du support des piles
# axcolor = 'lightgoldenrodyellow'
# ax_P = plt.axes([0.2, 0.1, 0.7, 0.03], axisbg=axcolor)  #xG,yG,long,epaisseur barre
jaune
# s_P = Slider(ax_P, 'P piles', 8, 50, valinit=P0)
# s_P.on_changed(update_P) #fonction appelée si mouvement slider
#
# #fonction associée au slider seuil:
# def update_seuil(val):
#     global seuil_perte
#     seuil_perte=s_seuil.val #lecture seuil sur le slider
#     X=np.arange(0,P)
#     Y=np.array(calcul_piles(P, seuil_perte,nb_actu0))
#     graph.set_xdata(X) #MAJ tracé
#     graph.set_ydata(Y) #MAJ tracé
#     Xcontours=np.arange(2*P)
#     Ycontours=np.arange(2*P)
#     for i in range(0, P):
#         Xcontours[2*i]=i
#         Xcontours[2*i+1]=i+1
#         Ycontours[2*i]=Y[i]
#         Ycontours[2*i+1]=Y[i]
#     graph_contours.set_xdata(Xcontours) #MAJ tracé
#     graph_contours.set_ydata(Ycontours) #MAJ tracé
```

```
# #slider seuil pertes
# axcolor = 'lightgoldenrodyellow'
# ax_seuil = plt.axes([0.2, 0.05, 0.7, 0.03], axisbg=axcolor)  #xG,yG,long,epaisseur
barre jaune
# s_seuil = Slider(ax_seuil, 'seuil pertes', 10, 2000, valinit=seuil_perte0)
# s_seuil.on_changed(update_seuil)

##TP8 partie 3
##euler
import numpy as np
import matplotlib.pyplot as plt # ça peut servir ...
#intiailisation des variables
Kt,Kn=1e-6,1e-6
m=10e-6
g=9.81
nu=0.5
gamma=5e-5

def F(X):
    return np.array([X[1],-Kt/m*X[0]-nu*g])

def G(Y):
    return np.array([Y[1],-Kn/m*Y[0]-gamma/m*Y[1]-g])

X1=np.array([0,0])
Y1= np.array([1e-5,0])

print(F(X1))
print(G(Y1))

#[  1.00000000e-03  -4.90500000e+00]
#[ 0.       -9.810001]
##
def euler(F,G, a, b, n):
    h = (b - a) / n
    T=np.linspace(a,b,n)
    X = np.zeros((n,2))
    Y = np.zeros((n,2))
    X[0]=X1
    Y[0]=Y1
    for k in range(n-1):
        X[k+1] = X[k]+h*F(X[k])
        Y[k+1] = Y[k]+h*G(Y[k])
    return T, X, Y

T,X,Y=euler(F,G, 0, 200, 200000)
print("temps",T)
print("x", X)
print("y",Y)


#mettre des titres aux axes et au graphique
plt.subplot(1,2,1)
plt.xlabel ('temps')
plt.ylabel ('deplacements')
#legende et correspondances des couleurs
plt.plot (T,X[:,0],'b',label=u'x(t)')
plt.legend(loc=1)

plt.subplot(1,2,2)
plt.xlabel ('temps')
#legende et correspondances des couleurs
plt.plot (T,Y[:,0],'r',label=u'y(t)')
plt.legend(loc=1)

##
plt.figure(2)
```

```
plt.subplot(1,2,1)
plt.xlabel ('x')
plt.ylabel ('y')
#legende et correspondances des couleurs
plt.plot (X[:,0],Y[:,0],'b',label=u'x en fonction de y ')
plt.legend(loc=1)

plt.subplot(1,2,2)
plt.xlabel ('y')
#portrait de phase
plt.plot (Y[:,0],Y[:,1],'r',label=u"y' en fonction de y")
plt.legend(loc=1)
```