```python
### TD2 PT etudes des piles et files
### Exercice 1 : le marchand de sable

### question 1 fonctions élémentaires des piles pas d'info sur la longueur
def creer_pile():
    return []

def empiler(pile,element):
    return pile.append(element)

def depiler(pile):
    return pile.pop()

def est_vide(pile):
    return len(pile)==0

### question 2 taille de la pile /// donc on ne la connait pas

def taille_pile(pile):
    pile_a_remplir=creer_pile()
    i=0
    while not est_vide(pile):
        empiler(pile_a_remplir,depiler(pile))
        i+=1
    while not est_vide(pile_a_remplir): #boucle qui reforme la pile initiale
        empiler(pile,depiler(pile_a_remplir))
    return (i) #préciser la question pile est modifiée (effet de bord)

### question 3 empiler des étoiles
def empilerSable(pile):
    return pile.append('*')

### gestion des tas de sable
### question 4 creer un tas de n piles de sable

def creer_tas(n):
    tas=[]
    for i in range(n):
        tas.append(creer_pile())
    return (tas)


### question 5 dépiler un grain de sable de la pile n et empiler la pile n-1
tas_sable=[[], ['*'], ['*','*','*'], ['*'], ['*','*']]
if not est_vide(tas_sable[2]):
    depiler(tas_sable[2])
    empilerSable(tas_sable[1])

# >>> tas_sable
# [[], ['*', '*'], ['*', '*'], ['*'], ['*', '*']]


# >>> creer_tas(5)
# [[], [], [], [], []]

import random as rd
### ecoulement
### question 6
# if taille_pile(tas[i])==taille_pile(tas[i+1]) and taille_pile(tas[i])>taille_pile(tas[i-1]):
#     sens=0
#
# ### question 7
# if taille_pile(tas[i])>taille_pile(tas[i+1]) and taille_pile(tas[i])>taille_pile(tas[i-1]):
#     sens=rd.randint(0,1)
#
# ### question 8
# if taille_pile(tas[i])==taille_pile(tas[i+1]) and
taille_pile(tas[i])==taille_pile(tas[i-1]):
#     sens=None

### question 9 bord du sablier
```

```
# if indice==0 or indice==n-1:
#     sens==None

### question 10 fonction chute
def chute(tas,indice,sens):
    empilerSable(tas[indice])
    if indice==0 or indice==n-1 or (taille_pile(tas[indice])==taille_pile(tas[indice+1]) and
    taille_pile(tas[indice])==taille_pile(tas[indice-1])):
        sens=None
    elif taille_pile(tas[indice])==taille_pile(tas[indice+1]) and
    taille_pile(tas[indice])>taille_pile(tas[indice-1]):
        sens=0
    elif taille_pile(tas[indice])>taille_pile(tas[indice+1]) and
    taille_pile(tas[indice])>taille_pile(tas[indice-1]):
        sens=rd.randint(0,1)
    else:
        sens=1
    if sens==None:
        return tas
    if sens==0:
        if not est_vide(tas[indice]):
            grain=depiler(tas[indice])
        return chute(tas,indice-1,0)
    if sens==1:
        if not est_vide(tas[indice]):
            grain=depiler(tas[indice])
        return chute(tas,indice+1,1)


### question 11 affichage du tas
def affichage(tas):
    n=taille_pile(tas)
    if n%2==0:
        for i in range(n//2+1):
            print ('-'*(n//2-i)+'*'*(i*2)+'-'*(n//2-i))
    else:
        for i in range(n//2+2):
            if i==0:
                print ('-'*(n//2-i)+'-'+'-'*(n//2-i))
            else:
                print ('-'*(n//2+1-i)+'*'*(i*2-1)+'-'*(n//2+1-i))

# >>> affichage(creer_tas(10))
# ----------
# ----**----
# ---****---
# --******--
# -********-
# **********

### Exercice 2 : notation polonaise inversee avec l'utilisation de listes
# est-ce un element au hasard ?

### question 1
def est_nombre(pile,i):
    return pile[i] not in ['+','-','*','/']

# >>> est_nombre(('+','1','3','*'),1)
# True

### question 2
def est_operation(pile,i):
    return pile[i] in ['+','-','*','/']

# >>> est_operation(('+','1','3','*'),0)
# True
# >>> est_operation(('+','1','3','*'),1)
# False

### question 3 inverser les éléments de la pile
def inversion(pile):
```

```
    pile_copie=pile[:] #ne pas modifier la pile de départ, il faut faire une copie
    pileInv=creer_pile()
    while not est_vide(pile_copie):
        empiler(pileInv,depiler(pile_copie))
    return (pileInv)

### question 4 évaluer l'expression postfixée on travaille sur une liste
def evaluer(exp):
    """la liste exp doit commencer par deux valeurs numériques, cette fonction renvoie la
    valeur calculée"""
    exp2=inversion(exp) # pour travailler sur le dépilement de la pile mais pas
    obligatoirement nécessaire
    reserve=[]
    if est_nombre(exp2,-1):
        reserve.append(depiler(exp2))
        if est_nombre(exp2,-1):
            reserve.append(depiler(exp2))
        else:
            print ('Erreur dans l\'expression, elle doit commencer par 2 nombres')
            return None
    for i in range(2,len(exp)):
        element=depiler(exp2)
        if element=='+':
            b,a=depiler(reserve),depiler(reserve)
            reserve.append(a+b)
        elif element=='-':
            b,a=depiler(reserve),depiler(reserve)
            reserve.append(a-b)
        elif element=='*':
            b,a=depiler(reserve),depiler(reserve)
            reserve.append(a*b)
        elif element=='/':
            b,a=depiler(reserve),depiler(reserve)
            reserve.append(a/b)
        else:
            reserve.append(element)
    return depiler(reserve)

#evaluer([1,2,'+',4,'*',3,'-',5,'+'])
#14

# evaluer([1,2,'+',3,10,'*','*',4,'+'])
# 94

### recursivité avec arbre binaire
from decimal import Decimal,getcontext

getcontext().prec = 5

#Classe utilisée pour stocker les noeuds de l'arbre
class Node:
    maxValue = 0
    maxDepth = 0
    nodeNumber = 0
    def __init__(self,data,par):
        self.data = data
        self.par = par
        self.right = None
        self.left = None
        if data:
            Node.nodeNumber += 1
        self.isnb = str(data).isdigit()
        if data and self.isnb and self.data > Node.maxValue:
            Node.maxValue = self.data

#Fonction (récursive) pour créer l'arbre
#Paramètres : objet Node, int (parcours de chaine), int (calcul de la profondeur max)
#Renvoie : tuple(Node,int) avec le noeud traité et l'index en cours
def makeTree(n, ind, depth):
    if depth > Node.maxDepth:
        Node.maxDepth = depth
```

```
    if ind < len(datas) and not datas[ind].isdigit() and not n.left:
        n.left,ind = makeTree(Node(datas[ind],n),ind + 1,depth + 1)
    if ind < len(datas) and datas[ind].isdigit() and not n.left:
        n.left = Node(int(datas[ind]),n)
        ind += 1
    if ind < len(datas) and datas[ind].isdigit() and not n.right:
        n.right = Node(int(datas[ind]),n)
        ind += 1
    if ind < len(datas) and not datas[ind].isdigit() and not n.right:
        n.right,ind = makeTree(Node(datas[ind],n),ind + 1,depth + 1)
    return n,ind

def calc(a,b,s):
    if s == "+":
        return a+b
    elif s == "-":
        return a-b
    elif s in "x":
        return a*b
    elif s == "/":
        return Decimal(a)/Decimal(b)

#Fonction (récursive) qui calcul le résultat total
#Paramètre : Node (noeud parent pour commencer)
#Renvoie : int (valeur du noeud en cours)
def calcResult(n):
    if not n.left.isnb and not n.right.isnb:
        return calc(calcResult(n.left),calcResult(n.right),n.data)
    elif not n.left.isnb and n.right.isnb:
        return calc(calcResult(n.left),n.right.data,n.data)
    elif n.left.isnb and not n.right.isnb:
        return calc(n.left.data,calcResult(n.right),n.data)
    else:
        return calc(n.left.data,n.right.data,n.data)
```