1 Présentation

2 Création des objets

2.1 Gestion des piles de sables

Une pile de sable est modélisée par... une pile dont la taille n'est pas limitée. Cette dernière est implémentée sous forme d'une liste.

Question 1 *Donner l'implémentation des fonctions élémentaires permettant de créer une pile dans Python à savoir les fonctions* `creer_pile`, `empiler`, `depiler`, `est_vide`. ***Pour cette question on s'autorise l'utilisation des méthodes sur les listes.***

Correction

```
def creer_pile():
    return []

def empiler2(pile,el):
    pile.append(el)

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

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

Question 2 *Donner l'implémentation de la fonction* `taille_pile`, *permettant de connaître la taille d'une pile.* ***Pour cette question seules les fonctions définies ci-dessus sont acceptées.*** *La pile ne doit pas être modifiée à l'issue de la fonction. Donner la complexité algorithmique de la fonction implémentée. La fonction devra être commentée.*

Correction

```
def taille(pile):
    i=0
    pile_tmp=creer_pile()
    while not est_vide(pile):
        i=i+1
        empiler(pile_tmp)
        depiler(pile)
    while not est_vide(pile_tmp):
        depiler(pile_tmp)
        empiler(pile)
    return i
```

La complexité de l'opération est une $\mathcal{O}(n)$.

Question 3 *Redéfinir la fonction* `empiler` *pour que le seul élément empilable soit la chaîne de caractères "*". Ainsi, une pile de sable sera constituée d'une pile d'étoiles.*

Correction

```
def empiler(pile):
    pile.append("*")
```

2.2 Gestion du tas de sable

Un tas de sable va être modélisé par une liste de piles de grains de sable.

Question 4 *Implémenter la fonction* `creer_tas` *permettant de créer un tas de* `n` *piles (vides et verticales) de sable.*

Correction

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

Question 5 *Donner l'(ou les) instruction(s) permettant de dépiler un grain de sable sur la pile n et d'empiler un grain de sable sur la pile $n-1$.*

Correction

```
depiler(tas[n])
empiler(tas[n-1])
```

3 Écoulement

Question 6 *Exprimer la condition booléenne pour laquelle un grain de sable chute à gauche.*

Correction `(taille(tas[indice_m-1])==hauteur_tas-2 and taille(tas[indice_m+1])==hauteur_tas-1)`

Question 7 *Exprimer la condition booléenne pour laquelle un grain de sable chute aléatoirement à gauche ou à droite.*

Correction `(taille(tas[indice_m-1])==hauteur_tas-2 and taille(tas[indice_m+1])==hauteur_tas-2)`

Question 8 *En réalisant un schéma donner un cas de figure pour lequel il n'y a pas d'écoulement de grain. Traduire la condition booléenne correspondante.*

Correction On ne fait rien si on n'est pas dans les cas précédents ...

On considère un écoulement vers la gauche.

Question 9 *Exprimer la condition booléenne permettant de savoir si un grain sur la pile n doit s'écouler sur la gauche. On tiendra compte du cas ou le grain est sur le bord du sablier. On pourra s'appuyer sur des figures*

Correction `(taille(tas[indice])>taille(tas[indice-1])+1 and taille(tas[indice])>1 and indice >0)`

On appelle `cond` la condition exprimée à la question précédente. On appelle `chute` la fonction permettant de régir la chute du grain (**ici vers la gauche**). Les spécifications de la fonction sont les suivantes :

▪ Python
```
def chute(tas,indice,sens):
    """
    Gestion d'une chute de grain de sable.
    Entrées :
     * tas(liste de piles) : tas de sable
     * indice(int) : pile àlaquelle on a déposé le dernier grain de sable
     * sens(int) : 0 chute àgauche, 1 chute àdroite
    Sortie :
     * le tas est modifié mais n'est pas retourné.
    """
```

Question 10 *Implémenter la fonction* `chute` *permettant de gérer la chute d'un grain de sable vers la gauche.* ***Cette fonction devra être récursive.*** *La fonction devra être commentée.*

Correction

```
def chute(tas,indice,sens):
    if sens == 0 :
        if (taille(tas[indice])>taille(tas[indice-1])+1 and taille(tas[indice])>1 and indice >0):
            depiler(tas[indice])
            empiler(tas[indice-1])
            chute(tas,indice-1,sens)
        else :
            return None
```


4 Affichage du tas de sable

Question 11 *Implémenter la fonction* `affichage` *permettant d'afficher un tas sous la forme définie ci-dessus.*

Correction

```
def affichage_tas(tas,hauteur_tas):

    for i in range(hauteur_tas,0,-1):
        ch=""
        for pile in tas :
            if taille(pile)>=i:
                ch = ch +"*"
            else :
                ch=ch + "_"
        print(ch)
```

Question 12 *Question subsidiaire – Créer la fonction* `sablier` *prenant comme argument la taille de la base du sablier et le nombre de grains à écouler. Cette fonction devra retourner un tas de sable résultat de l'écoulement d'un seul grain.*

Correction

```
def chute(tas,indice,sens):
    """
    Gestion d'une chute de grain de sable.
    Entrées :
     * tas(liste de piles) : tas de sable
     * indice(int) : pile à laquelle on a déposé le dernier grain de sable
     * sens(int) : 0 chute à gauche, 1 chute à droite
    Sortie :
     * le tas est modifié mais n'est pas retourné.
    """
    if sens == 0 :
        if (taille(tas[indice])>taille(tas[indice-1])+1 and taille(tas[indice])>1 and indice >0):
            depiler(tas[indice])
            empiler(tas[indice-1])
            chute(tas,indice-1,sens)
        else :
            return None
    if sens == 1 :
        if (indice==len(tas)-1):
            return None
        elif (taille(tas[indice])>taille(tas[indice+1])+1 and taille(tas[indice])>1):
            depiler(tas[indice])
            empiler(tas[indice+1])
            chute(tas,indice+1,sens)
        else :
            return None
    return None
```