Exercice 3 : Tas de sable

Le but de cet exercice est de définir une fonction permettant de calculer le résultat de l'éboulement d'un tas de sable. On considère ici que les tas de sable sont constitués de plusieurs colonnes de grains de sable côte à côte. Chaque colonne est elle-même constituée d'un certain nombre de grains de sable.

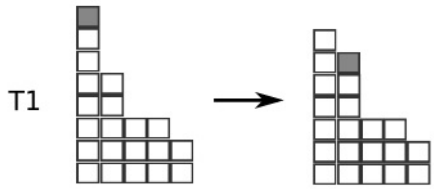
On choisit donc de représenter un tas de sable par une liste d'entiers naturels, chaque entier représentant le nombre de grains dans une colonne.
Soit les deux exemples de tas de sable `T1` et `T2` suivant :

Le tas `T1` sera donc représenté par la liste `(8 5 3 3 2)` et `T2` par la liste `(3)`.

Dans la suite, on supposera toujours que **les colonnes sont rangées par hauteur décroissante**. Autrement dit la liste d'entiers qui représente le tas est décroissante (au sens large). Il ne sera pas nécessaire de rappeler cette condition en hypothèse des fonctions de l'exercice.

Un éboulement est possible dans un tas si les hauteurs de deux colonnes adjacentes diffèrent de 2 grains de sable ou plus. Un éboulement n'est pas possible à partir de la dernière colonne.

Si un éboulement est possible entre deux colonnes de sable, celui-ci a pour effet de transférer un grain de sable de la colonne la plus haute (celle de gauche) vers la colonne la plus basse (celle de droite). En reprenant l'exemple du tas `T1`, le premier éboulement possible est le suivant :



Le grain en haut de la première colonne (en gris) tombe sur la deuxième colonne

Question 3.1 : [3/61]

Donner une définition avec signature et hypothèse(s) éventuelle(s) des fonctions suivantes :

- `eboult_possible` qui, étant donné un tas contenant au moins une colonne de sable, teste si un éboulement est possible.
- `indice_eboult` qui, étant donné un tas contenant au moins un éboulement possible, renvoie l'indice de la première colonne permettant un éboulement.

Par exemple :

```
>>> eboult_possible([3])
False
>>> eboult_possible([8, 7, 7, 6, 5])
False
>>> eboult_possible([8, 5, 3, 3, 2])
True
>>> indice_eboult([8, 5, 3, 3, 2])
0
```

```
>>> eboult_possible([8, 7, 6, 3, 2])
True
>>> indice_eboult([8, 7, 6, 3, 2])
2
```

Remarque : la totalité des points ne sera allouée qu'aux fonctions qui contiennent des sorties anticipées.

In [2]:
```python
from typing import List

def eboult_possible(L : List[int]) -> bool:
    """Precondition : len(L) >= 1
    Retourne True si un éboulement est possible dans le tas L"""
    ind_tas_courant : int = 0
```

```
    while ind_tas_courant < len(L)-1:
        if L[ind_tas_courant] - L[ind_tas_courant + 1] >= 2:
            return True
        ind_tas_courant = ind_tas_courant + 1
    return False

def eboult_possible(L : List[int]) -> bool:
    """Precondition : len(L) >= 1
    Retourne True si un éboulement est possible dans le tas L"""
    ind_tas_courant : int = 0
    for ind_tas_courant in range(len(L)-1):
        if L[ind_tas_courant] - L[ind_tas_courant + 1] >= 2:
            return True
    return False
```

In [3]:
```
def indice_eboult(L : List[int]) -> int:
    """Preconditions : len(L) >= 1
    Un éboulement doit être possible
    Retourne l'indice au niveau duquel un éboulement est possible"""
    ind_tas_courant : int = 0
    while ind_tas_courant < len(L)-1:
        if L[ind_tas_courant] - L[ind_tas_courant + 1] >= 2:
            return ind_tas_courant
        ind_tas_courant = ind_tas_courant + 1
    return ind_tas_courant

def indice_eboult(L : List[int]) -> int:
    """Preconditions : len(L) >= 1
    Un éboulement doit être possible
    Retourne l'indice au niveau duquel un éboulement est possible"""
    ind_tas_courant : int = 0
    for ind_tas_courant in range(len(L)-1):
        if L[ind_tas_courant] - L[ind_tas_courant + 1] >= 2:
            return ind_tas_courant
    return ind_tas_courant
```

In [ ]:
```
```

Question 3.2 : [3/61]

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `eboult_col` qui, étant donné un tas `T` et un indice `i` correspondant à un éboulement valide, réalise l'éboulement de la colonne `i` dans `T`.

Par exemple :

```
>>> eboult_col([8, 5, 3, 3, 2], 0)
[7, 6, 3, 3, 2]
>>> eboult_col([8, 7, 1], 1)
[8, 6, 2]
```

In [4]:
```
def eboul_col(L : List[int], ind : int) -> List[int]:
    """Precondition : len(L) >= 1
    L'indice correspond à un éboulement valide : L[ind] - L[ind+1] >= 2
    Retourne le tas après réalisation de l'éboulement"""
    # ma liste résultat
    L_resultat : List[int]
    L_resultat = L[0:ind] + [L[ind]-1, L[ind+1]+1] + L[ind+2:]
    return L_resultat
```

In [5]:
```
eboul_col([2,3,4,5,9,2,1,5,4,6], 4)
```

Out[5]:
```
[2, 3, 4, 5, 8, 3, 1, 5, 4, 6]
```

Question 3.3 : [3/61]

Un tas est dit *stable* s'il n'y a plus d'éboulement possible. En vous servant des fonctions précédentes, donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `stabilisation` qui, étant donné un tas contenant au moins une colonne de sable, réalise les éboulements successifs dans le tas jusqu'à ce que le tas soit stabilisé.

Par exemple :

```
>>> stabilisation([3])
[3]
>>> stabilisation([8, 5, 3, 3, 2])
[6, 5, 4, 3, 3]
>>> stabilisation([8, 7, 1])
[6, 5, 5]
```

In [6]:
```
def stabilisation(L : List[int]) -> List[int]:
    """Precondition : len(L) >= 1
    Retourne le tas après application des éboulements successifs. On obtient un tas stabilisé"""
    # indice où un éboulement est possible
    indice : int = 0
    # Tas résultat après réalisation de l'éboulement
    L_resultat : List[int] = L
    while eboult_possible(L_resultat):
        indice = indice_eboult(L_resultat)
        L_resultat = eboul_col(L_resultat, indice)
    return L_resultat
```

In [7]:
```
stabilisation([3])
```

Out[7]:
```
[3]
```

In [8]:
```
stabilisation([8,5,3,3,2])
```

Out[8]:
```
[6, 5, 4, 3, 3]
```

In [9]:
```
stabilisation([8,7,1])
```

Out[9]:
```
[6, 5, 5]
```

In [10]:
```
L=[12,15,4,8,2,4,688,254,226]
#   0  1 2 3 4 5  6   7   8
#   -9   -8  -7  -6  -5  -4   -3    -2    -1
L2 = ["salut", "ok", "cava", "bien"]
#         0      1      2       3
#        -4     -3      -2      -1
```

In [14]:
```
assert L[6] == 688
assert L[-7] == 4
assert L2[2] == "cava"
assert L2[-3] == "ok"
```

In [17]:
```
indice : int = 0
while indice < len(L):
    print(L[indice])
    indice = indice + 1
```

```
12
15
4
8
2
4
688
254
226
```

In [18]:
```
indice : int = 0
for indice in range(len(L)):
    print(L[indice])
```

```
12
15
4
8
2
4
688
254
226
```

In [19]:
```
i : int
for i in range(len(L)):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
```

Exercice 1 : Polynômes

Le but de cet exercice est de définir un ensemble de fonctions capables de manipuler des polynômes.

Un polynôme à une variable est un objet mathématique que l'on définit comme étant une somme de termes : $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$. Chaque terme du polynôme est de la forme a_kx^k dans lequel x est la variable du polynôme, k le *degré* du terme et a_k son coefficient.

Ainsi, le polynôme $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ est complètement défini par la liste des coefficients $[a_0, a_1, \ldots, a_n]$. Par exemple, le polynôme $P_1(x) = 5 - 2x + x^2$ est déterminé par la liste de ses coefficients `[5, -2, 1]`. De même, le polynôme $P_2(x) = 1 + x^4$ est représenté par `[1, 0, 0, 0, 1]` et le polynôme $P_3(x) = -x^4$ est représenté par `[0, 0, 0, 0, -1]`. Par ailleurs, on décide de représenter le polynôme nul par la liste `[0]`.

On s'intéresse, dans cet exercice, aux polynômes à coefficients entiers et nous pourrons utiliser l'alias de type `Poly` pour `list[int]`.

On supposera également dans la suite que les polynômes donnés en exemple ci-dessus ont été définis par les variables suivantes :

```
>>> poly1 = [5, -2, 1]
>>> poly2 = [1, 0 , 0, 0, 1]
>>> poly3 = [0, 0, 0, 0, -1]
>>> polynul = [0]
```

Remarque : Dans tout ce qui suit, nous pourrons supposer, sans le rajouter en hypothèse dans les spécifications, que tout polynôme contient au moins 1 coefficient, c'est à dire que la liste qui le représente est non vide.

Question 1.1 : [3/66]

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `poly_applique` qui, étant donné un polynôme `P` et un nombre `x`, renvoie la valeur du polynôme `P` en `x`.

Par exemple :

```
>>> poly_applique(poly1, 0)
5.0                          # P1(0) = 5 - 2*0 + 0^2 = 0
>>> poly_applique(poly1, 1)
4.0                          # P1(1) = 5 - 2*1 + 1^2 = 4
>>> poly_applique(poly1, 2)
5.0                          # P1(2) = 5 - 2*2 + 2^2 = 5
>>> poly_applique(poly3, 2)
-16.0                        # P3(2) = -2^4 = -16
>>> poly_applique(polynul, 2)
0.0
```

In [ ]:
```
```

Question 1.2 : [3/66]

On rappelle que l'addition de deux polynômes P et P' est donné par le polynôme P'' dont chacun des coefficients correspond à l'addition des coefficients des termes de même degré dans P et P'.

Attention, les degrés des deux polynômes ne sont pas forcément égaux. Si $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ et $P'(x) = b_0 + b_1x + b_2x^2 + \ldots + b_px^p$ avec $p > n$, alors l'addition de P et P' donne le polynôme $P''(x) = (a_0 + b_0) + (a_1 + b_1)x + (a_2 + b_2)x^2 + \ldots + (a_n + b_n)x^n + b_{n+1}x^{n+1} + \ldots + b_px^p$

Ainsi l'addition de P_1 et P_2 de l'exercice précédent donne le polynôme P suivant :

$$
\begin{array}{r|cccccc}
P_1 & 5 & -2 & 1 & & & \\
+ \quad P_2 & 1 & 0 & 0 & 0 & 1 &
\end{array}
$$

X^2	1	0	0	0	1
P	6	−2	1	0	1

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `poly_add` qui renvoie l'addition de deux polynômes. Par exemple :

```
>>> poly_add(poly1, poly2)
[6, -2, 1, 0, 1]
>>> poly_add(poly1, polynul)
[5, -2, 1]
>>> poly_add(poly2, poly3)
[1, 0, 0, 0, 0]
```

In [ ]: