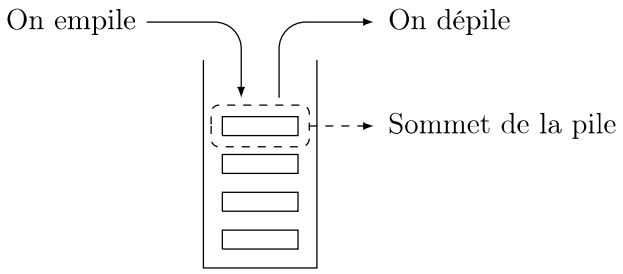
Exercice 1

Cet exercice porte sur la notion de pile et sur la programmation de base en Python.

On rappelle qu'une pile est une structure de données abstraite fondée sur le principe « dernier arrivé, premier sorti » :



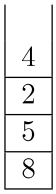
Une pile

On munit la structure de données Pile de quatre fonctions primitives définies dans le tableau ci-dessous. :

Structure de données abstraite : Pile
Utilise : Éléments, Booléen

Opérations :
— `creer_pile_vide` : $\emptyset \longrightarrow$ `Pile`
`creer_pile_vide()` renvoie une pile vide
— `est_vide : Pile` $\longrightarrow$ `Booléen`
`est_vide(pile)` renvoie True si `pile` est vide, False sinon
— `empiler : Pile, Élément` $\longrightarrow$ `Rien`
`empiler(pile, element)` ajoute `element` au sommet de la `pile`
— `depiler : Pile` $\longrightarrow$ `Élément`
`depiler(pile)` renvoie l'élément au sommet de la `pile` en le retirant de la `pile`

Question 1 On suppose dans cette question que le contenu de la pile `P` est le suivant (les éléments étant empilés par le haut) :



Quel sera le contenu de la pile `Q` après exécution de la suite d'instructions suivante ?

```
1       Q = creer_pile_vide()
2       while not est_vide(P):
3           empiler(Q, depiler(P))
```

Question 2

1. On appelle *hauteur* d'une pile le nombre d'éléments qu'elle contient. La fonction `hauteur_pile` prend en paramètre une pile `P` et renvoie sa hauteur. Après appel de cette fonction, la pile `P` doit avoir retrouvé son état d'origine.

   Exemple : si `P` est la pile de la question 1 : `hauteur_pile(P) = 4`.

   Recopier et compléter sur votre copie le programme Python suivant implémentant la fonction `hauteur_pile` en remplaçant les `???` par les bonnes instructions.

```
def hauteur_pile(P):
    Q = creer_pile_vide()
    n = 0
    while not(est_vide(P)):
        ???
        x = depiler(P)
        empiler(Q,x)
    while not(est_vide(Q)):
        ???
        empiler(P, x)
    return ???
```

2. Créer une fonction `max_pile` ayant pour paramètres une pile `P` et un entier `i`. Cette fonction renvoie la position `j` de l'élément maximum parmi les `i` derniers éléments empilés de la pile `P`. Après appel de cette fonction, la pile `P` devra avoir retrouvé son état d'origine. La position du sommet de la pile est `1`.

   Exemple : si `P` est la pile de la question 1 : `max_pile(P, 2) = 1`

Question 3 Créer une fonction `retourner` ayant pour paramètres une pile `P` et un entier `j`. Cette fonction inverse l'ordre des `j` derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

Exemple : si `P` est la pile de laquestion 1(a), après l'appel de `retourner(P, 3)`, l'état de la pile `P` sera :

Question 4 L'objectif de cette question est de trier une pile de crêpes.

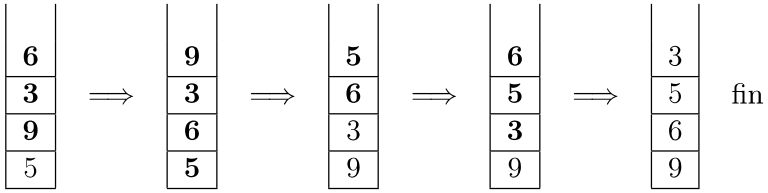
On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).

On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus.

Le principe est le suivant :

- On recherche la plus grande crêpe.
- On retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile.
- On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas.
- La plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

Exemple :



Créer la fonction `tri_crepes` ayant pour paramètre une pile `P`. Cette fonction trie la pile `P` selon la méthode du tri crêpes et ne renvoie rien. On utilisera les fonctions créées dans les questions précédentes.

Exemple : Si la pile `P` est

7
14
12
5
8

, après l'appel de `tri_crepes(P)`, la pile `P` devient

5
7
8
12
14

.