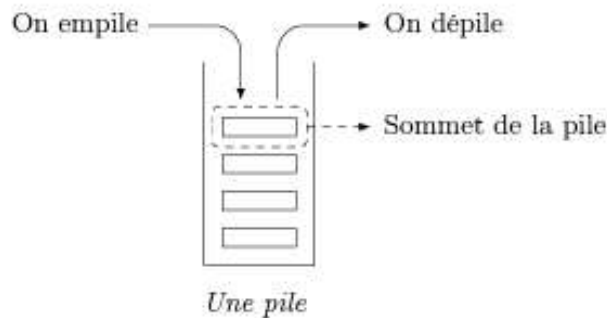


# Files, vers le bac

## 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 » :



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 :
— <code>creer_pile_vide</code> : $\emptyset \rightarrow$ Pile <code>creer_pile_vide()</code> renvoie une pile vide
— <code>est_vide</code> : Pile $\rightarrow$ Booléen <code>est_vide(pile)</code> renvoie True si <code>pile</code> est vide, False sinon
— <code>empiler</code> : Pile, Élément $\rightarrow$ Rien <code>empiler(pile, element)</code> ajoute <code>element</code> au sommet de la pile
— <code>depiler</code> : Pile $\rightarrow$ Élément <code>depiler(pile)</code> 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) :

4
2
5
8

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.

```
1     def hauteur_pile(P):
2         Q = creer_pile_vide()
3         n = 0
4         while not(est_vide(P)):
5             ???
6             x = depiler(P)
7             empiler(Q,x)
8         while not(est_vide(Q)):
9             ???
10            empiler(P, x)
11        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 la question 1(a), après l'appel de `retourner(P, 3)`, l'état de la pile `P` sera :

5
2
4
8

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

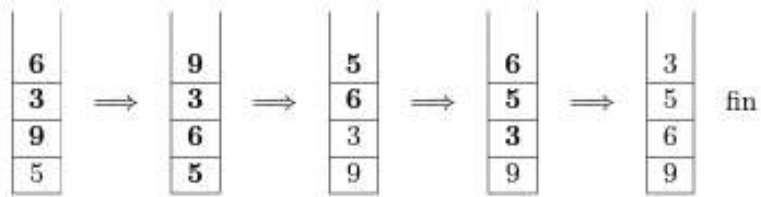
On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).

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.

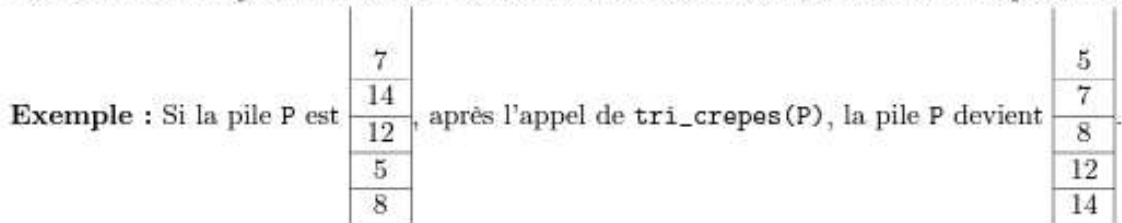
Le principe est le suivant :

- 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.



## Exercice 2

Cet exercice porte sur les structures de données linéaires

Une méthode simple pour gérer l'ordonnancement des processus est d'exécuter les processus en une seule fois et dans leur ordre d'arrivée.

1. Parmi les propositions suivantes, quelle est la structure de données la plus appropriée pour mettre en œuvre le mode FIFO (First In First Out) ?
  - a) liste
  - b) dictionnaire
  - c) pile
  - d) file
2. On choisit de stocker les données des processus en attente à l'aide d'une liste Python `lst`. On dispose déjà d'une fonction `retirer(lst)` qui renvoie l'élément `lst[0]` puis le supprime de la liste `lst`. Écrire en Python le code d'une fonction `ajouter(lst, proc)` qui ajoute à la fin de la liste `lst` le nouveau processus en attente `proc`.

On choisit maintenant d'implémenter une file `file` à l'aide d'un couple  $(p1, p2)$  où `p1` et `p2` sont des piles. Ainsi `file[0]` et `file[1]` sont respectivement les piles `p1` et `p2`.

Pour enfiler un nouvel élément `elt` dans `file`, on l'empile dans `p1`.

Pour défiler `file`, deux cas se présentent.

- La pile `p2` n'est pas vide : on dépile `p2`.
- La pile `p2` est vide : on dépile les éléments de `p1` en les empilant dans `p2` jusqu'à ce que `p1` soit vide, puis on dépile `p2`.

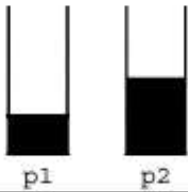
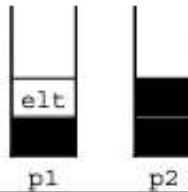
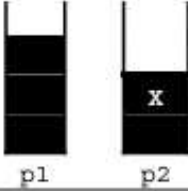
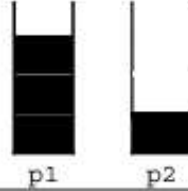
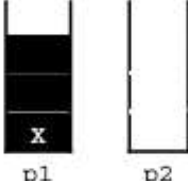
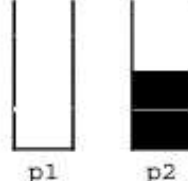
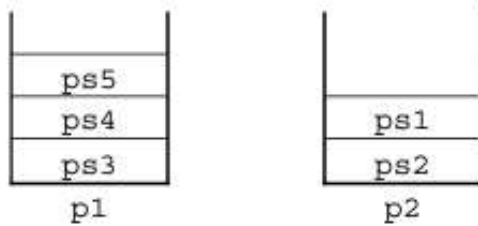
	État de la file avant	État de la file après
<code>enfiler(file, elt)</code>		
<code>defiler(file)</code> cas où <code>p2</code> n'est pas vide		
<code>defiler(file)</code> cas où <code>p2</code> est vide		

Illustration du fonctionnement des fonctions `enfiler` et `defiler`.

3. On considère la situation représentée ci-dessous.



On exécute la séquence d'instructions suivante :

```
enfiler(file,ps6)
defiler(file)
defiler(file)
defiler(file)
enfiler(file,ps7)
```

Représenter le contenu final des deux piles à la suite de ces instructions.

4. On dispose des fonctions :

- `empiler(p,elt)` qui empile l'élément `elt` dans la pile `p`,
- `depiler(p)` qui renvoie le sommet de la pile `p` si `p` n'est pas vide et le supprime,
- `pile_vide(p)` qui renvoie `True` si la pile `p` est vide, `False` si la pile `p` n'est pas vide.

a. Écrire en Python une fonction `est_vide(f)` qui prend en argument un couple de piles `f` et qui renvoie `True` si la file représentée par `f` est vide, `False` sinon.

b. Écrire en Python une fonction `enfiler(f,elt)` qui prend en arguments un couple de piles `f` et un élément `elt` et qui ajoute `elt` en queue de la file représentée par `f`.

c. Écrire en Python une fonction `defiler(f)` qui prend en argument un couple de piles `f` et qui renvoie l'élément en tête de la file représentée par `f` en le retirant.

### Exercice 3

Cet exercice porte sur les structures de données (listes, piles et files).

On cherche ici à mettre en place des algorithmes qui permettent de modifier l'ordre des informations contenues dans une file. On considère pour cela les structures de données abstraites de Pile et File définies par leurs fonctions primitives suivantes :

#### **Pile :**

- `creer_pile_vide()` renvoie une pile vide ;
- `est_pile_vide(p)` renvoie `True` si la pile `p` est vide, `False` sinon ;
- `empiler(p, element)` ajoute `element` au sommet de la pile `p` ;
- `depiler(p)` renvoie l'élément se situant au sommet de la pile `p` en le retirant de la pile `p` ;
- `sommet(p)` renvoie l'élément se situant au sommet de la pile `p` sans le retirer de la pile `p`.

#### **File :**

- `creer_file_vide()` renvoie une file vide ;
- `est_file_vide(f)` renvoie `True` si la file `f` est vide, `False` sinon ;
- `enfiler(f, element)` ajoute `element` dans la file `f` ;
- `defiler(f)` renvoie l'élément à la tête de la file `f` en le retirant de la file `f`.

On considère de plus que l'on dispose d'une fonction permettant de connaître le nombre d'éléments d'une file :

- `taille_file(f)` renvoie le nombre d'éléments de la file `f`.

On représentera les files par des éléments en ligne, l'élément de droite étant la tête de la file et l'élément de gauche étant la queue de la file. On représentera les piles en colonnes, le sommet de la pile étant le haut de la colonne.

La file suivante est appelée `f` :

4	3	8	2	1
---	---	---	---	---

La pile suivante est appelée `p` :

5
8
6
2

- Les quatre questions suivantes sont indépendantes. Pour chaque question, on repartira de la pile `p` et de la file `f` initiales (présentées ci-dessus)

a. Représenter la file `f` après l'exécution du code suivant.

```
enfiler(f, defiler(f))
```

b. Représenter la pile `p` après l'exécution du code suivant.

```
empiler(p, depiler(p))
```

c. Représenter la pile `p` et la file `f` après l'exécution du code suivant.

```
for i in range(2):  
    enfiler(f, depiler(p))
```

d. Représenter la pile  $p$  et la file  $f$  après l'exécution du code suivant.

```
for i in range(2):
    empiler(p, defiler(f))
```

2. On donne ici une fonction `mystere` qui prend une file en argument, qui modifie cette file, mais qui ne renvoie rien.

```
def mystere(f):
    p = creer_pile_vide()
    while not est_file_vide(f):
        empiler(p, defiler(f))
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
    return p
```

Préciser l'état de la variable  $f$  après chaque boucle de la fonction

`mystere` appliquée à la file 

1	2	3	4
---	---	---	---

 Indiquer le contenu de la pile renvoyée par la fonction.

3. On considère la fonction `knuth(f)` suivante dont le paramètre est une file :

```
def knuth(f):
    p=creer_pile_vide()
    N=taille_file(f)
    for i in range(N):
        if est_pile_vide(p):
            empiler(p, defiler(f))
        else:
            e = defiler(f)
            if e >= sommet(p):
                empiler(p, e)
            else:
                while not est_pile_vide(p) and e < sommet(p):
                    enfiler(f, depiler(p))
                empiler(p, e)
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
```

a. Recopier et compléter le tableau ci-dessous qui détaille le fonctionnement de cet algorithme étape par étape pour la file 2, 1, 3. Une étape correspond à une modification de la pile ou de la file. Le nombre de colonnes peut bien sûr être modifié.

$f$	2,1,3	2,1										
$p$		3										

b. Que fait cet algorithme ?