Les files

Exercice 1 :

À la caisse du supermarché, mamie (qui fait ses courses le samedi) apporte un panier bien garni.

Un par un, elle dépose minutieusement chacun de ses achats.

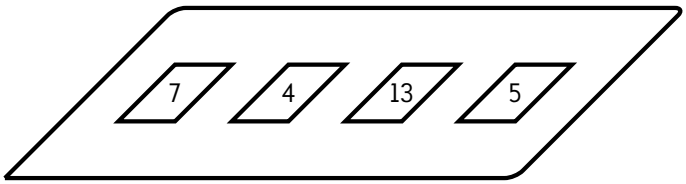
On modélise le tapis de caisse par une liste où le dernier élément correspond au premier article ajouté. Lorsque mamie n'a pas commencé, le tapis est vide et la liste aussi.

À la caisse, il est possible d'effectuer quatre actions :

- ❏ Créer une caisse (en la mettant en service) ;
- ❏ Vérifier si le tapis de caisse est vide ;
- ❏ Ajouter un article sur le tapis (par mamie) ;
- ❏ Retirer un article du tapis (par l'hôte de caisse) ;

Par soucis de simplification, on dira qu'un article est représenté par un entier.

Par exemple, la liste `[7, 4, 13, 5]` représente le tapis de caisse :



En python, les quatre actions possibles pour une caisse sont représentées par les fonctions suivantes :

Action	Fonction
Créer une caisse	`creer_caisse() -> list`
Vérifier si le tapis est vide	`est_vide(L : list) -> bool`
Ajouter un article au tapis	`ajouter_article(L : list, n : int) -> None`
Retirer un article au tapis	`retirer_article(L : list) -> int`

1. Quelles ont été les instructions pour générer le tapis de caisse `[7, 4, 13, 5]` ?

2. Mamie souhaite récupérer l'article numéro 4. Quelles instructions doit-elle effectuer ?

3. Écrire une implantation des quatre actions possibles pour une caisse.

D.W.

Cours

En informatique, une **FILE** est une structure de données abstraite qui respecte le principe suivant :

Premier arrivé = Premier sorti

On résume ce principe par l'acronyme **FIFO** pour **First In, First Out**.
Cette structure de données peut être modélisée avec différents objets en Python :

- ❏ Une liste ;
- ❏ Un dictionnaire à deux entrées : `"tête"` et `"reste"` ;
- ❏ Une classe (Voir le chapitre sur la programmation objet) ;

L'**interface** de cette structure est constituée des primitives suivantes :

nom	argument(s)	description
creer_file	vide	Crée une file vide
est_vide	File f	Vérifie si la file f est vide
enfiler	File p, valeur v	Enfile la valeur v dans la file f
defiler	File f	Retire la valeur en tête de file (si possible) et la renvoie

Exercice 2 :

On considère le programme suivant :

```
file = creer_file()
enfiler(file, 0)
enfiler(file, 1)
for _ in range(10):
    a = defiler(file)
    b = defiler(file)
    enfiler(file, b)
    enfiler(file, a+b)
```

1. Donner l'état de la file après chaque tour de boucle.

Exercice 3 :

Dans cet exercice, on propose l'implantation suivante de la primitive `creer_file` :

```
def creer_file() -> dict:
    """ Crée une pile version dictionnaire """
    res = {"tête" : None, "reste" : []}
    return res
```

Par exemple, la file $\rightarrow \boxed{8 \quad 3 \quad 5 \quad 1} \rightarrow$ est représentée par le dictionnaire

```
{"tête" : 1, "reste" : [8, 3, 5]}
```

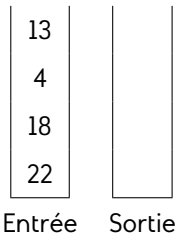
1. Écrire l'implantation de `enfiler(F : dict, n : int) -> None` qui enfile l'entier n dans la file F.
2. Écrire l'implantation de `defiler(F : dict) -> int` qui défile la tête de F.

Exercice 4 :

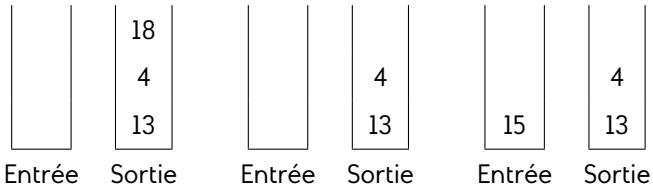
Pour modéliser une file, il est possible d'utiliser deux piles : une pile d'entrée, une pile de sortie.

- ❏ Pour enfiler une valeur, on l'empile dans la pile d'entrée ;
- ❏ Pour défiler une valeur, deux cas sont possibles :
  - — La pile de sortie est vide : on déverse la pile d'entrée dans la pile de sortie ;
  - — La pile de sortie n'est pas vide, on dépile cette dernière.

On possède la file $\rightarrow \boxed{13 \quad 4 \quad 18 \quad 22} \rightarrow$. L'état des piles Entrée et Sortie est donné par :



Les instructions `defiler(f)`, `defiler(f)` et `enfiler(f, 15)` produisent :



La file a été modélisée par un tuple nommé grâce au code :

```
from collections import namedtuple

File = namedtuple('File', ['entree', 'sortie'])
```

où `'entree'` et `'sortie'` sont des identifiants représentants des piles.

Remarque : On rappelle que pour accéder aux attributs `'entree'` et `'sortie'`, on doit utiliser les instructions `f.entree` et `f.sortie` où `f` a été initialisée par `f = File([], [])`

1. Implémenter la fonction `est_vide` pour une file `f`.

2. Implémenter la fonction `enfiler`.

3. Recopier et compléter le code suivant qui implémente la fonction `defiler` :

```
def defiler(f):
    if ... :
        res = None
    else:
        if est_vide(...):
            while not est_vide(...):
                empiler(..., depiler(...))

        res = depiler(f.sortie)
    return res
```