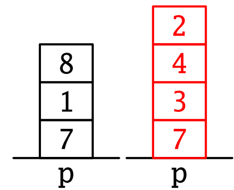
Exercice 1 : On part avec une pile représentée ci-contre. Représenter l'état de la pile après les instructions suivantes :

```
>>> depiler(p)
>>> depiler(p)
>>> empiler(3, p)
```

```
>>> empiler(5, p)
>>> depiler(p)
>>> empiler(0, p)
```

```
>>> depiler(p)
>>> empiler(4, p)
>>> empiler(2, p)
```

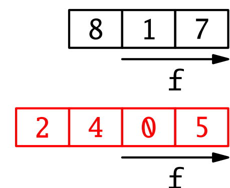


Exercice 2 : On part avec une file représentée ci-contre. Représenter l'état de la file après les instructions suivantes :

```
>>> retirer(f)
>>> retirer(f)
>>> ajouter(3, f)
```

```
>>> ajouter(5, f)
>>> retirer(f)
>>> ajouter(0, f)
```

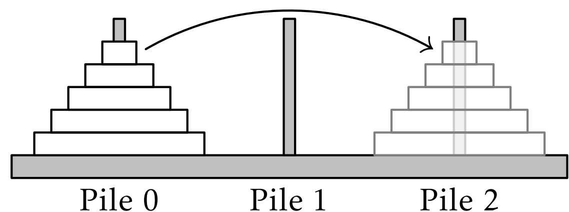
```
>>> retirer(f)
>>> ajouter(4, f)
>>> ajouter(2, f)
```



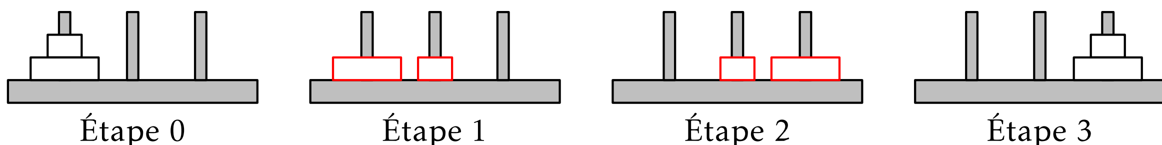
Les tours de Hanoï

On appelle **tours de Hanoï** un jeu, inventé par le mathématicien Édouard Lucas, composé de disques de bois s'empilant sur 3 tiges. L'objectif est de déplacer vers la droite la pile de disques se trouvant à gauche, et cela en un minimum de coups. Pour cela il faut respecter les règles suivantes :

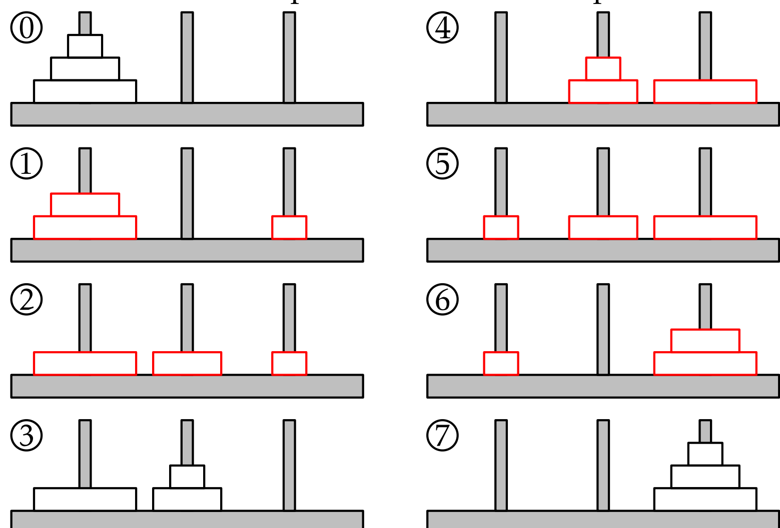
- On ne peut déplacer qu'un disque à la fois.
- On ne peut déplacer qu'un disque sur un autre disque plus grand que lui ou sur un emplacement vide.
- Dans la position initiale, l'empilement de disques respecte la règle précédente.



Exercice 3 : Il faut 3 étapes pour déplacer une pile de 2 disques. Compléter les étapes intermédiaires sur le schéma ci-dessous :



Exercice 4 : Il faut 7 étapes pour déplacer une pile de 3 disques. Compléter les étapes manquantes sur le schéma ci-contre :



Programmation des tours de Hanoï

Tous les exercices de cette partie ont pour but de programmer le jeu et de permettre de le résoudre automatiquement. Pour cela on dispose de deux interfaces pour les piles : une avec une structure non spécifiée et des fonctions, et une autre avec une approche objet. Pour les exercices, vous pouvez choisir d'utiliser l'approche de votre choix.

Fonction	Méthode de POO	Description
`creer_pile()`	`creer_pile()`	Renvoie une pile vide.
`empiler(pile, element)`	`pile.empiler(element)`	Empile au sommet.
`depiler(pile)`	`pile.depiler()`	Renvoie et enlève la valeur au sommet.
`est_vide(pile)`	`pile.est_vide()`	Renvoie un booléen.

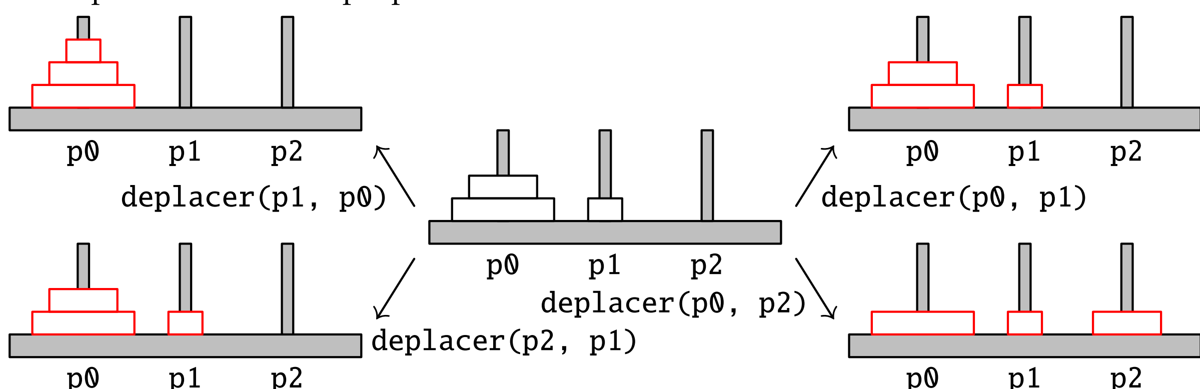
Dans tous les cas, les piles sont modélisées par des objets mutables. Cela veut dire que toute modification faite lors de l'exécution d'une fonction perdurera après. Il n'est donc pas nécessaire d'utiliser de `return` pour renvoyer le nouvel état de la pile.
Vous pouvez utiliser la fonction d'un exercice dans les exercices suivants.
Dans la suite, on notera `p0`, `p1` et `p2` les piles sur les 3 tiges, de gauche à droite.

Exercice 5 : Écrire une fonction `sommet(pile)` qui renvoie la valeur au sommet de la pile. Vous pouvez dépiler, mais il faut bien rempiler la valeur avant la fin de l'exécution de la fonction. Si la pile est vide, il faut lever une exception avec `raise IndexError('pile vide')`.

```
def sommet(pile):
    if est_vide(pile):
        raise IndexError('pile vide')
    else:
        v = depiler(pile)
        empiler(pile, v)
        return v
```

Exercice 6 : On souhaite écrire une fonction `deplacer(origine, cible)` qui déplace la valeur au sommet de la pile `origine` vers le sommet de la pile `cible`. Si le déplacement n'est pas possible, parce qu'il ne respecte pas les règles du jeu, les piles ne sont pas modifiées.

1) Dans chacun des cas ci-dessous, en partant de l'état central, dessiner l'état des piles attendu après l'instruction proposée.



2) Écrire le code de la fonction `deplacer(origine, cible)` :

```
def deplacer(origine, cible):
    if not est_vide(origine):
        if est_vide(cible) or sommet(origine) < sommet(cible):
            v = depiler(origine)
            empiler(origine, v)
```

Exercice 7 : Donner la suite d'instructions nécessaires pour passer de l'étape 0 à l'étape 3 dans l'exercice 3, en utilisant la fonction `deplacer`.

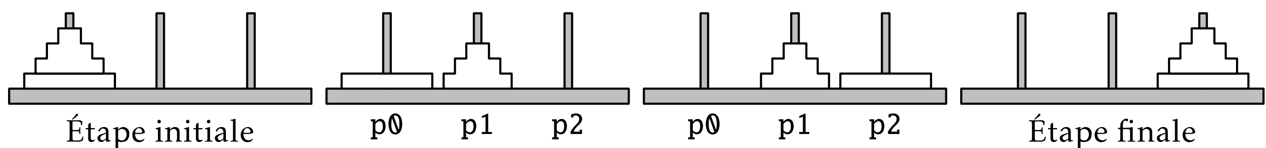
```
deplacer(p0, p1)
deplacer(p0, p2)
deplacer(p1, p2)
```

Exercice 8 : On veut écrire une fonction récursive `resoudre(n, origine, cible, interm)` qui permet de déplacer les `n` premiers disques au sommet de la pile `origine` vers la pile `cible`, en utilisant éventuellement la pile `interm` comme pile intermédiaire pour les déplacements.

1) Indiquer la commande à effectuer si `n == 1`.

```
deplacer(origine, cible)
```

2) On considère maintenant une pile de `n` disques sur la pile `p0` qu'on souhaite amener en `p2`, avec $n > 1$. En utilisant `resoudre(n-1, ..., ..., ...)` pour déplacer les $n-1$ premiers disques et `deplacer` pour le dernier disque, donner les instructions permettant d'effectuer les étapes ci-dessous :




```
resoudre(n-1, p0, p1, p2)
deplacer(p0, p2)
resoudre(n-1, p1, p2, p0)
```

Exercice 9 : On souhaite écrire une fonction récursive `nb_etapes(n)` qui renvoie le nombre d'étapes nécessaires pour déplacer une pile de `n` disques, avec $n > 1$. Vous pouvez vous inspirer de l'exercice précédent.

1) Combien vaut `nb_etapes(1)` ? `nb_etapes(1)` = 1

2) Exprimer `nb_etapes(n)` en fonction de `nb_etapes(n-1)`. C'est à dire s'il faut `nb_etapes(n-1)` étapes pour déplacer une pile de `n-1` disques, combien en faut-il pour déplacer une pile de `n` disques ? `nb_etapes(n)` = $2 \times$ `nb_etapes(n-1)` $+ 1$

3) Compléter le code de la fonction :

```
def nb_etapes(n):
    if n == 1:
        return 1
    else:
        return 2 * nb_etapes(n-1) + 1
```