Structure de données abstraites / Exercices

1. Exercice 1 : Choisir un TAD selon la situation

Q1. Choisir le type de donnée abstraite adéquate pour chacune des tâches suivantes :

- ➔ Stocker un répertoire téléphonique
- ➔ Stocker l'historique de navigation d'un navigateur web
- ➔ Stocker les fichiers à imprimer dans un serveur d'impression
- ➔ Stocker les cartes d'un joueur dans le jeu de la bataille

2. Exercice 2 : Utiliser l'interface d'un TAD

Q1. Pour chaque séquence de primitives suivantes, **représenter** par un croquis l'état du TAD à chaque étape

Séquence 1

```
l = creer_liste()
inserer(l, 'A',1)
inserer(l, 'O',2)
inserer(l, 'B',1)
inserer(l, 'V',3)
inserer(l, 'R',2)
```

Séquence 2

```
f = creer_file()
enfiler(f,4)
enfiler(f,1)
enfiler(f,3)
n = defiler(f)
enfiler(f,8)
n = defiler(f)
```

Séquence 3

```
l1 = creer_liste()
l2 = creer_liste()
inserer(l1, 1, 1)
inserer(l1, 2, 2)
inserer(l1, 3, 3)
inserer(l1, 4, 4)
inserer(l2, lire(l1, 1), 1)
inserer(l2, lire(l1, 2), 1)
inserer(l2, lire(l1, 3), 1)
inserer(l2, lire(l1, 4), 1)
```

Séquence 4

```
p = creer_pile()
empiler(p,4)
empiler(p,3)
n = depiler(p)
empiler(p,8)
n = depiler(p)
```

3. Exercice 3 : Raisonner avec un TAD

On souhaite développer un nouveau TAD appelé ***2extremites*** ayant la capacité d'insérer et retirer un élément des deux extrémités. L'interface de ce TAD est le suivant :

Nom	Opérations
`creer_2extremites()`	Retourne un *2extremites* vide
`2_est_vide(t)`	Teste si **t** est vide. Renvoie True si vide.
`inserer_gauche(t,e)`	Insère à gauche de **t** un élément **e**
`retirer_gauche(t)`	Renvoie l'élément situé à gauche de **t**
`inserer_droite(t,e)`	Insère à droite de **t** un élément **e**
`retirer_droite(t)`	Renvoie l'élément situé à droite de **t**

Cette nouvelle structure sera implémentée avec une pile principale associée à une pile secondaire qui permettra de renverser la première si besoin. La primitive `inserer_droite(t,e)` revient à empiler un élément dans la pile principale. Par contre pour `inserer_gauche(t,e)` il faut dépiler l'ensemble des éléments de la pile principale dans la pile secondaire puis empiler le nouvel élément dans la pile principale vide et enfin dépiler l'ensemble des éléments de la pile secondaire dans la pile principale.

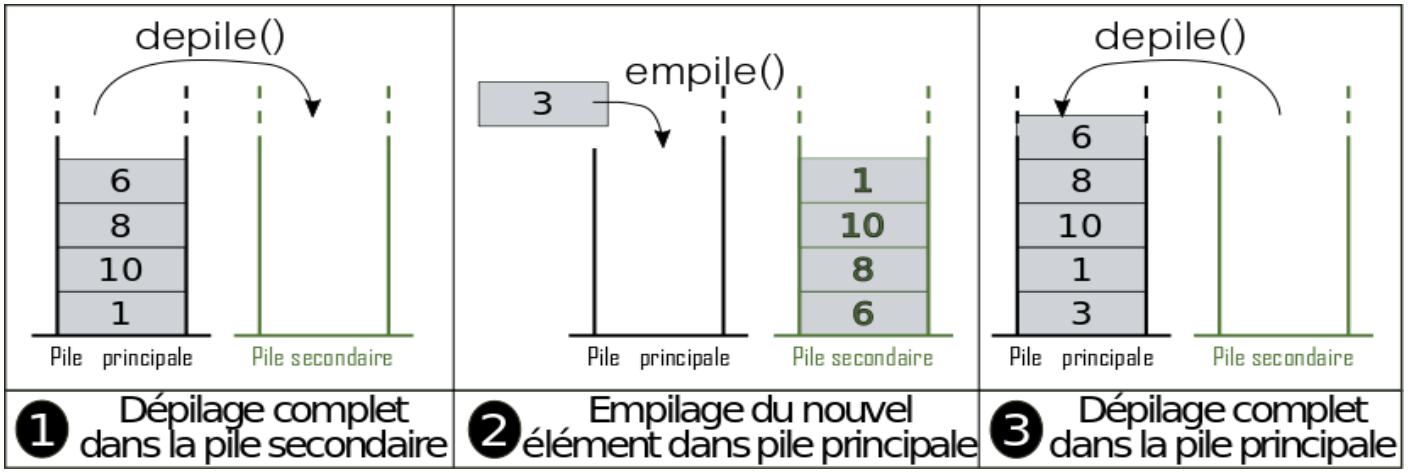

Figure 1: Illustration de la primitive **`inserer_gauche(t,e)`**

Ces deux primitives ainsi que le constructeur du TAD peuvent s'écrire :

```
fonction creer_2extremites() :
     retourner creer_pile()

fonction inserer_droite(t, e)
     empiler(t, e)
```

```
fonction inserer_gauche(t, e) :
     t_secondaire = creer_pile()
     tant que non(est_vide(t)) :
          empiler(t_secondaire, depiler(t))
     empiler(t, e)
     tant que non(est_vide(t_secondaire)) :
          empiler(t, depiler(t_secondaire))
```

Q1. Définir les primitives 2**`_est_vide(t)`** et **`retirer_droite(t)`**
Q2. Illustrer l'évolution des 2 piles (principale et secondaire) lors de l'appel de **`retirer_gauche(t)`**.
Q3. Définir la primitive **`retirer_gauche(t)`**

4.Exercice 4 : Implémentation d'une pile avec un tableau

On souhaite définir une classe Pile implémentant une pile à base d'un tableau.
Le tableau utilisé peut contenir n éléments :

- ➔ la première case du tableau (indice 0) contient l'indice de la prochaine case vide (emplacement correspondant à l'indice où sera inséré le prochain élément)
- ➔ les cases suivantes du tableau (indices 1 à n), contiennent les éléments de la pile ou sont vides. La dernière case non vide correspond au sommet de la pile

D'après cette description, on en déduit que :

- ➔ si P[0]==1 → la pile est vide.
- ➔ A chaque fois que l'on insère un élément, on augment P[0] d'une unité
- ➔ Lorsque P[0] == n+1, la pile est pleine

Le constructeur de la classe Pile est la suivante :

```
class Pile :
    def __init__(self)
    self.__n = 20
    self.__p = [None] * self.__n
    self.__p[0] = 1
```

Q4. Définir dans cette classe les méthode **`empiler()`**, **`depiler()`** et **`est_vide()`** en respectant l'interface du TAD pile défini dans le cours.

Q5. Valider le fonctionnement de cette classe en vérifiant les résultats obtenus dans l'exercice 2 - séquence 4

5.Exercice 5 : Implémentation d'une file avec une pile

On souhaite maintenant définir une classe File implémentant une file à base d'une pile. Cette pile sera une une instance de la classe Pile définie dans l'exercice 4.

Q6. A partir de la classe Pile et des raisonnements réalisés dans l'exercice 3, **définir** une classe File respectant l'interface d'une file présentée dans le cours.

Q7. Valider le fonctionnement de cette classe en vérifiant les résultats obtenus dans l'exercice 2 - séquence 1