# TP n° 4 : les piles

## 1. Manipulations de base des piles

### 1.1. Présentation

Soit une pile $P$ composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (le sommet de la pile est 22).
Pour chaque exemple ci-dessous on repart de la pile d'origine $P$.

- *depiler(P)*. Renvoie 22 et la pile $P$ est maintenant composée des éléments suivants : 12, 14, 8, 7 et 19 (le sommet de la pile est 19).
- *empiler(P, 42)*. La pile P est maintenant composée des éléments suivants : 12, 14, 8, 7, 19, 22 et 42.
- *sommet(P)*. Renvoie 22, la pile P n'est pas modifiée.
- Si on applique *depiler(P)* 6 fois de suite, alors pile_vide(P) renvoie vrai.
- Si on applique *depiler(P)* une fois, alors *taille(P)* renvoie 5.

### 1.2. Exercice

Soit une pile $P$ composée des éléments suivants : 12, 8, 5, 23 et 30 (le sommet de la pile est 30).
Décrire l'effet sur la pile à chaque étape. On part, cette fois, de l'état de la pile de l'état précédent.

1. *depiler(P)*
2. *sommet(P)*
3. *empiler(P, 18)*
4. *taille(P)*
5. *empiler(P, 42), depiler(P), depiler(P), taille(P)*

## 2. Implémentation en utilisant des fonctions

### 2.1. Présentation

Dans python, la méthode la plus simple pour gérer une pile, consiste à utiliser un tableau et ses fonctions associées pour stocker les éléments de la pile.

1. Quel est le rôle de la méthode *append()* sur un tableau en Python ?
2. Même question pour la méthode *pop()*.
3. Même question pour la fonction *len()*.
4. Que renvoie l'instruction *tab[-1]* si *tab* est un tableau non vide ?

**Remarque :** la méthode *pop()* renvoie une erreur si elle est appliquée sur un tableau vide.

**Remarque :** même si on utilise des tableau pour implémenter le TAD pile en Python, il faut avoir conscience que certaines opérations associées aux tableau ne sont pas des opérations de piles. Par exemple, faire un accès à n'importe lequel de ses éléments n'est pas une opération de pile.

### 2.2. Exercice

L'objectif est de créer une pile en utilisant uniquement un tableau, ainsi que les méthodes *append()* et *pop()* sans indice. En particulier, la fonction *len()* ne devra pas être utilisée.

Mis à part avec les méthodes *creer_pile()*, *empile()* et *depile()*, la pile ne devra pas être modifiée. Il faut parfois la régénérer à la fin si la méthode l'a modifiée.

Ouvrir le programme *TP04_piles_fonctions.py*. Remplacer progressivement les instructions *pass* pour implémenter les fonctions suivantes :

- *creer_pile()* fonction qui retourne un tableau vide.
- *empile(p, a)* fonction qui ajoute l'élément $a$ au tableau $p$.
- *depile(p)* fonction qui supprime le dernier élément du tableau $p$.
- *est_vide(p)* fonction qui retourne *True* si le tableau est vide.
- *sommet(p)* fonction qui retourne le dernier élément du tableau.

- *hauteur(p)* fonction qui renvoie la taille (hauteur) de la pile.

Le programme pourra être testé avec les instructions suivantes :

```
if __name__ == '__main__':
    # tester les fonctions
    p = creer_pile()
    empile(p, 1)
    empile(p, 2)
    empile(p, 3)
    empile(p, 4)
    print(p)
    echange_haut_bas(p)
    print(p)
```

S'il y a le temps, créer une fonction *echange_haut_bas()* qui échange le sommet et le fond de la pile *p*.

## 3. Implémentation en POO

### Objectif

Créer une classe d'objet Pile qui simule les fonctionnalités d'une pile avec un tableau.

### Cahier des charges

Ouvrir le programme *TP04_piles_classe.py*. La classe *Pile* contiendra les méthodes :

- *empile()* en utilisant la méthode *append()* ;
- *depile()* en utilisant la méthode *pop()* ;
- *est_vide()*
- *sommet()*
- *hauteur()* en utilisant la fonction *len()*.

### Test

- Créer une pile *p*, puis empiler successivement les valeurs 3 puis 5 et enfin 10.
- Afficher cette pile *p*.
- Afficher la hauteur de cette pile.
- Dépiler la premier valeur, puis afficher *p*.
- Recommencer pour les deux valeurs suivantes.

## Capacités exigibles

- Écrire plusieurs implémentations d'une même structure de données.