Chapitre 5 - Les piles

Objectifs :

- ▷ Distinguer des structures par le jeu des méthodes qui les caractérisent.
- ▷ Choisir une structure de données adaptée à la situation à modéliser.
- ▷ Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.

1 Introduction

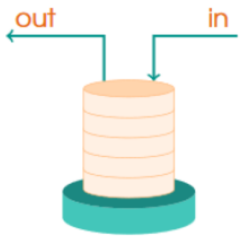
Dans ce chapitre nous allons décrire des **structures de données linéaires** appelées **piles**. Il faut bien comprendre que lorsqu'on parle de structure de données, on parle d'une représentation **abstraite** qui n'est pas en lien direct avec son implémentation qui peut être réalisée de diverses manières suivant le langage de programmation, voire au sein d'un même langage de programmation.

2 Définition

A retenir !

En informatique, une pile (*stack* en anglais) est une structure de données fondée sur le principe du **dernier arrivé, premier sorti** (ou **LIFO** pour *Last In, First Out*).

Cela signifie que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.



Le fonctionnement est donc celui d'une **pile d'assiettes** : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée.

3 Exemples d'usage

Voici quelques exemples d'usage courant d'une pile :

- ▷ Dans un navigateur web, une pile sert à **mémoriser l'historique des pages web visitées**. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant le bouton *«Afficher la page précédente»*.
- ▷ L'évaluation des **expressions mathématiques en notation post-fixée** (ou polonaise inverse) utilise une pile.
- ▷ La fonction **"Annuler"** (en anglais *Undo*) d'un logiciel (photo, traitement de texte, ..., etc.) mémorise les dernières modifications effectuées dans une pile.

4 Implémentation

Pour implémenter une structure de pile, on a besoin d'implémenter seulement un nombre réduit d'opérations de bases qui sont :

- ▷ **empiler** : ajoute un élément sur la pile. Terme anglais correspondant : *push*
- ▷ **dépiler** : enlève un élément de la pile et le renvoie. En anglais : *pop*
- ▷ **vide** : renvoie vrai si la pile est vide, faux sinon
- ▷ **remplissage** : renvoie le nombre d'éléments dans la pile

La structure de pile est un **concept abstrait**. Comment faire pour réaliser une pile dans la pratique ?

L'idée principale étant que les fonctions de bases pourront être utilisées indépendamment de l'implémentation choisie.

4.1 Implémentation n°1

Nous utiliserons une simple liste pour représenter la pile. Il se trouve que les méthodes append et pop sur les listes jouent déjà le rôle de *push* et *pop* sur les piles.

Voici les fonctions de base :

```
def pile():
    ''' Retourne une liste vide'''
    return []

def vide(p):
    '''Renvoie True si la pile est vide et False sinon'''
    return p == []

def empiler(p,x):
    '''Ajoute l'element x a la pile p'''
    p.append(x)

def depiler(x):
    '''Depile et renvoie l'element au sommet de la pile p'''
    assert not vide(p), "Pile vide"
    return p.pop()
```

A faire 1 : Tester les instructions suivantes :

```
p = pile()

for i in range(5):
    empiler(p,2*i)

a = depiler(p)
print(a)
print(vide(p))
```

Expliquer les affichages obtenus :

A faire 2 : Réaliser les **fonctions** `taille(p)` et `sommet(p)` qui retournent respectivement la taille de la liste et le sommet de la liste (sans le supprimer).

4.2 Implémentation n°2

Nous allons utiliser la POO afin de créer une classe `Pile` pour implémenter cette structure.

```
class Pile:
    '''classe Pile, creation d'une instance Pile avec une liste'''

    def __init__(self):
        '''Initialisation d'une pile vide'''
        self.L = []

    def vide(self):
        '''Teste si la pile est vide'''
        return self/L == []

    def depiler(self):
        '''Depile la dernier element de la pile'''
        assert not self.vide(), 'Pile vide'
        return self.L.pop()

    def empiler(self,x):
        '''Empile l'element x en haut de la pile'''
        assert not self.vide(), 'Pile vide'
        return self.L.append(x)
```

Tester les instructions suivantes :

```
p = Pile()
for i in range(5):
    p.empiler(2*i)

print(p.L)
a = p.depiler()
print(a)

print(p.L)
print(p.vide())
```

A faire 3 : Réaliser les **méthodes** `taille(self)` et `sommet(self)` qui retournent respectivement la taille de la liste et le sommet de la liste (sans le supprimer).

5 Exercices

https ://glassus.github.io/terminale$_n$si/T1$_S$tructures$_d$e$_d$onnees/1.1$_L$istes$_P$iles$_F$iles/cours/
Exercice 1 :