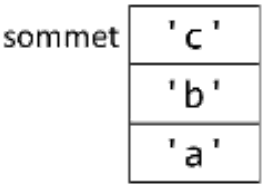
Structures linéaires : Piles

Cours

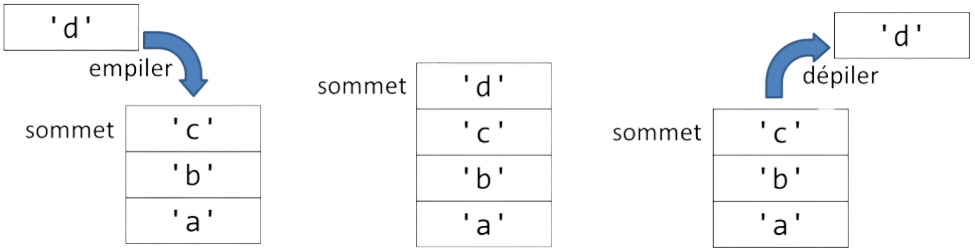
En informatique, une **pile** (en anglais ***stack***) est un type abstrait de données sur le principe « dernier arrivé, premier sorti » (ou **LIFO** pour ***Last In, First Out***).

Le dernier élément ajouté à la pile, ou **empilé**, est le premier qui sera sorti, ou **dépilé**.

Le dernier élément empilé est le **sommet** de la pile.



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.



Les piles trouvent de nombreuses applications en informatique, par exemple :

- Dans un navigateur web, une pile sert à mémoriser les 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 ».
- La fonction « Annuler la frappe » (en anglais « Undo ») d'un traitement de texte mémorise les modifications apportées au texte dans une pile.

Interface

Les principales primitives constituant l'interface d'une pile sont :

- `creer() → pile` : construire une pile vide.
- `est_vide() → bool` : vérifier si une pile est vide ou non.
- `empiler(element)` : ajouter un élément sur la pile (*Push* en anglais).
- `dépiler() → element` : enlèver un élément de la pile et le renvoyer (*Pop* en anglais).
- `taille() → int` : renvoyer le nombre d'éléments dans la pile (la hauteur).

Exemple :

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 : - `dépiler(P)` renvoie 22 et la pile `P` est maintenant composée des éléments suivants : 12, 14, 8, 7 et 19 (le sommet de `P` est 19). - `empiler(P, 42)` la pile `P` est maintenant composée des éléments suivants : 12, 14, 8, 7, 19, 22 et 42 - si on applique `dépiler(P)` 6 fois de suite, `est_vide(P)` renvoie vrai. - après avoir appliqué `dépiler(P)` une fois, `taille(P)` renvoie 5.

Implémentation

La pile est un type abstrait, son implémentation peut se faire sous différentes formes, par exemple avec une liste chaînée ou un tableau dynamique de type `list` Python.

Avec une liste chaînée

Sur la même idée que la classe `ListeChainee` vue précédemment, il est possible de peut créer une classe `Pile` de toute pièce basée sur la class `Cellule`.

```
class Cellule:
    '''Cellule de Pile '''

    def __init__(self, v, d=None):
        self.valeur = v      # Valeur de la Cellule
        self.dessous = d     # Cellule de dessous

class Pile:
    '''Pile sous forme d'une liste chainee de cellules'''
    def __init__(self):
        self.sommet = None
        self._hauteur = 0          # Nombre d'élément de la Pile

    def est_vide(self):
        return self.sommet == None

    def empiler(self, v):
        self.sommet = Cellule(v, self.sommet)
        self._hauteur += 1

    def depiler(self):
        if self.est_vide(): raise IndexError("la pile est vide")
        v = self.sommet.valeur
        self.sommet = self.sommet.dessous
        self._hauteur -= 1
        return v

    def taille(self):
        return self._hauteur
```

Créons maintenant une instance de `Pile` :

```
>>> p = Pile()
>>> p.taille()
0
>>> p.empiler('a')
>>> p.empiler('b')
>>> p.empiler('c')
```

```
>>> p.depiler()
'c'
>>> p.taille()
2
```

Avec le type `list` de Python

Il est aussi possible de créer très facilement une classe `Pile` en utilisant le type `list` avec les méthodes `pop()` et `append()` :

```
class Pile:
    ''' structure de Pile en utilisant le type list'''

    def __init__(self):
        self.pile = []

    def est_vide(self):
        return len(self.pile) == 0

    def empiler(self, v):
        self.pile.append(v)

    def depiler(self):
        if self.est_vide(): raise IndexError("la pile est vide")
        return self.pile.pop()

    def taille(self):
        return len(self.pile)

    def __str__(self):
        ch = ''
        for v in self.pile:
            ch = ch + "|\t" + str(v) + "\t|\n"   # Affiche les élements de la pile : |  v  |
        ch = ch + "---------"
        return ch
```

Créons maintenant une instance de `Pile` :

```
>>> p = Pile()
>>> p.est_vide()
True
>>> p.depiler()
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
  File "<module1>", line 15, in depiler
IndexError: la pile est vide
>>> p.empiler('a')
>>> p.empiler('b')
>>> p.empiler('c')
>>> p.est_vide()
False
>>> p.depiler()
'c''
>>>
```

Enregistons cette classe de `Pile` dans un fichier "pile.py", elle nous sera utile par la suite.

Pour faire encore plus simple, programmons une pile avec la même idée, mais sans utiliser la POO :

```
def creer():
    return  []

def est_vide(p):
    return len(p) == 0

def empiler(p, v):
    p.append(v)         # Inutile de renvoyer p, le type list est muable

def depiler(p):
    if est_vide(p): raise IndexError("la pile est vide")
    return p.pop()

def taille(p):
    return len(p)

def afficher(p):
    for v in p:
        print("|\t", v, "\t|\n")   # Affiche les élements de la pile : |  v  |
    print("---------")
```

puis

```
>>> p = creer()
>>> est_vide(p)
True
>>> depiler(p)
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
  File "<module1>", line 11, in depiler
IndexError: la pile est vide
>>> empiler(p, 'a')
>>> empiler(p, 'b')
>>> empiler(p, 'c')
>>> depiler(p)
'c'
>>> taille(p)
2
```

Finalement en Python, une simple variable de type `list` avec la fonction `len()` et les méthodes `.append()` et `.pop()` est une façon rapide et simple de créer une pile avec toutes ses primitives :

```
>>> p = []           # creer()
>>> len(p) == 0      # est_vide()
True
>>> p.append('a')    # empiler()
>>> p.append('b')    # empiler()
>>> p.append('c')    # empiler()
>>> len(p)           # taille()
3
>>> p.pop()          # depiler()
'c'
```