Une structure de données est une organisation logique des données permettant de simplifier ou d'accélérer leur traitement.

## 0.1 Introduction

En informatique, une pile (en anglais stack) est une structure de données fondée sur le prin- cipe *dernier arrivé, premier sorti* (ou LIFO pour Last In, First Out), ce qui veut dire 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. Voici quelques exemples d'usage courant d'une pile:

- 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*.
- L'évaluation des expressions mathématiques en notation post-fixée (ou polonaise inverse) utilise une pile.
- La fonction *Annuler la frappe* (en anglais *Undo*) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.

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

- `empiler` : ajoute un élément sur la pile. Terme anglais correspondant : `Push` .
- `depiler` : enlève un élément de la pile et le renvoie. En anglais : `Pop`.
- `est_vide` : renvoie vrai si la pile est vide, faux sinon.
- `taille` : renvoie le nombre d'éléments dans la pile.

La structure de pile est un concept abstrait. Comment la réaliser dans la pratique? Voici plusieurs implémentations possibles. L'idée principale étant que les fonctions de bases pourront être utilisées indépendamment de l'implémentation choisie.

## 0.2 Implémentation

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. Ecrire les fonctions de base :

```
def creer_pile():
"""
retourne une pile vide
"""
    pass

def vide(p):
"""
renvoie True si la pile est vide
et False sinon
"""
    pass

def empiler(p,x):
"""
Ajoute l'élément x à la pile p
"""
    pass
```

```
def depiler(p):
"""
dépile et renvoie l'élément au sommet de la pile p
"""
    pass
```

**Exercices**

**Exercice 1**

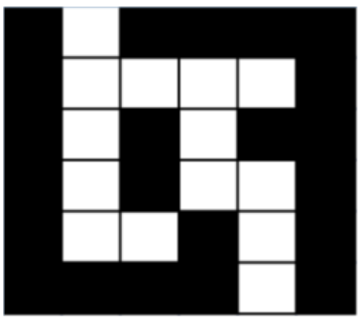
1. Ecrire une classe `Pile` avec son constructeur et les primitives suivantes: `est_vide()`; `depiler()`; `empiler()`.
2. Tester les instructions suivantes:

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

3. Réaliser les fonctions `taille(p)` et `sommet(p)` qui retourne respectivement la taille de la liste et le sommet de la liste ( sans le supprimer).

**Exercice 2** L'objectif est d'écrire un algorithme de parcours d'un labyrinthe, donnant le chemin de l'entrée vers la sortie.

On représentera le labyrinthe à l'aide d'un tableau (liste de listes).



```
laby = [[0,1,0,0,0,0],
        [0,1,1,1,1,0],
        [0,1,0,1,0,0],
        [0,1,0,1,1,0],
        [0,1,1,0,1,0],
        [0,0,0,0,1,0]]
```

On repère une case par ses coordonnées (i,j), on y accède dans le tableau par `laby[i][j]` L'entrée se fait par la case (0,1) et la sortie par la case (5,4).

**PARTIE A**

1. Ecrire une fonction `hauteur()` et une fonction `largeur()` renvoyant respectivement le nombre de lignes et le nombre de colonnes du tableau `tab` passé en paramètre.

```
lignes = len( )

colonnes = len( )
```

La fonction `afficher_grille` affiche une liste de liste.

2. **Objectif**: L'objectif est d'écrire un programme qui détermine s'il existe un chemin de l'entrée vers la sortie en se déplaçant vers le haut, le bas, la gauche ou la droite (mais pas en diagonale). Le tableau d'origine `laby` ne devra pas être modifié.

Ci-contre une fonction qui prend en paramètre un tableau T et un tuple v.

(a) Que renvoie cette fonction ?

(b) Expliquer l'affichage provoqué par cette instruction :

```
>>>print(mystere(laby,(0,1))
```

```
def mystere(T,v):
    V = []
    i, j = v[0],v[1]
    for a in (-1,1):
        if 0<=i+a<hauteur(T):
            if T[i+a][j]==1:
                V.append((i+a,j))
        if 0<=j+a<largeur(T):
            if T[i][j+a]==1:
                V.append((i,j+a))
    return V
```

**PARTIE B**

1. Proposer un algorithme de parcours de ce labyrinthe basé sur un type de donnée `pile`.
2. Compléter l'algorithme de la fonction `parcours(laby, entree, sortie)`

```
tab⟵copie de .....
p⟵ Pile()
v⟵ entree
case v dans tab⟵.....
recherche ⟵ Vrai
Tant Que recherche est Vraie
    vois ← voisins(....., .....)
    Si la liste ..... est vide alors
        Si p est vide alors
            retourner Faux
        sinon
            v ⟵.....
        FinSi
    sinon
        on empile p avec v
        v ⟵ 1er .....
        case ..... dans .....⟵.....
        Si v = ..... alors
            On ..... p avec .....
            recherche⟵ .....
    FinSi
FinTantQue
retourner p
```

## 0.3 Références


- Td Piles du lycée de Draguignan (Van-Zuijlen Stephan).
- Terminale NSI-Ellipses, Serge Bays.