**I**nstitut **P**réparatoire aux **E**tudes d'**I**ngénieurs de **N**abeul
**Informatique :** TP Python
**Niveau :** $2^{ème}$ années SP, ST et SM

# TP 1. Structures de données avancées : Pile et File

## La structure Pile

**Exercice 1 :**

Une `pile` est une structure de données de type `LIFO` (**L**ast **I**n **F**irst **O**ut) : le dernier entré est le premier sorti. Créer un module Python « `pile.py` » contenant les fonctions suivantes en essayant de fixer une taille maximale de la pile par l'introduction d'une constante NMAX=40. Prévoir les traitements nécessaires au bon fonctionnement du module en cas de saturation de la pile.

1. `creer_pile()` : permet de créer et renvoyer une pile vide ;
2. `est_vide(p)` : renvoie « True » si la pile « p » est vide, « False » sinon.
3. `sommet(p)` : renvoie l'élément du sommet de la pile « p ».
4. `depiler(p)` : supprime de la pile « p » le sommet.
5. `empiler(p,x)` : ajoute au sommet de la pile « p » l'élément « x ».

**Exercice 2 :**

En utilisant les fonctions définies dans le module « `pile.py` » de l'exercice précédent, écrivez les fonctions suivantes :

1. `afficher(f)` : permet d'afficher tous les éléments de la pile.
2. `depilerKelt(p,k)` : dépile « k » éléments si la pile contient au moins « k » éléments, sinon elle dépile toute la pile.
3. `depilerJusqua(p,elt)` : cette fonction dépile la pile jusqu'à l'élément « elt » et l'élément « elt » n'est pas dépilé. Si l'élément n'appartient pas à la pile, alors la fonction dépile toute la pile.

**Exercice 3 : conversion base 10 / base 2**

Écrire une fonction `conversion(nb10)` qui retourne la représentation en base 2 du nombre nb10 représenté en base 10 à l'aide de piles.

**Exercice 4 : Évaluation d'une expression arithmétique**

Une des tâches fondamentales d'un interpréteur comme celui de Python (ou d'un compilateur dans un autre langage de programmation) est d'attribuer une valeur à une expression arithmétique. Une expression arithmétique est formée des 10 chiffres $0, 1, \ldots, 9$, des signes $+$, $-$, $\times$ et $/$ et des parenthèses ouvrantes et fermantes. Par exemple : $(\ 3 + (4 \times 5)\ )/2$ est une expression arithmétique dont la valeur est 11,5. En général, l'utilisateur tape cette expression à l'aide de son clavier, ce qui génère une chaîne de caractères $ch =$ "(3 + (4 ∗ 5))/2". Le problème est de traduire cette chaîne en un nombre, tout en gérant les priorités des opérations et les parenthèses : l'interpréteur doit se livrer pour cela à une **analyse syntaxique**.
Une même expression arithmétique a au moins trois représentations naturelles :

1. La représentation **infixée**. C'est celle qu'on utilise tout le temps : $(3 + 2) \times 5$
2. La représentation **préfixée** où on place tous les opérateurs avant les opérandes : 2 + 5 s'écrit + 2 5 et $(3 + 2) \times 5$ s'écrit $\times$ + 3 2 5. Dans ce cas, les parenthèses sont superflues :
si on avait voulu dire : $3 + (2 \times 5)$, on aurait écrit : + 3 × 2 5.

3. On peut également placer les opérateurs après les opérandes : c'est la représentation **postfixée**. Dans ce cas 2 + 5 s'écrirait : 2 5 + et si on partait de (3 + 2) × 5 on aurait : 3 2 + 5 ×. Aucun besoin de parenthèses là non plus.

**Questions**

1. Donner les deux autres formes des expressions suivantes et indiquer à chaque fois leurs valeurs :
   a) 2 + ( 3 - (5 + 1) × 2 )
   b) 2 3 4 5 + - 6 × /

2. L'évaluation d'une expression **postfixée** peut se faire de façon très simple, en une seule lecture de la gauche vers la droite à l'aide d'une pile qui stocke les résultats intermédiaires. Sur l'exemple suivant, le caractère | a été rajouté pour indiquer la limite de la partie déjà traitée de l'expression.
   | 2 3 4 + - 5 × pile $P$ : []
   2 | 3 4 + - 5 × pile $P$ : [2]
   2 3 | 4 + - 5 × pile $P$ : [2,3]
   2 3 4 | + - 5 × pile $P$ : [2,3,4]
   2 3 4 + | - 5 × pile $P$ : [2,7]
   2 3 4 + - | 5 × pile $P$ : [−5]
   2 3 4 + - 5 | × pile $P$ : [−5,5]
   2 3 4 + - 5 × | pile $P$ : [−25] **Résultat :** -25
   Partons de la chaîne postfixée : $ch$ = "123 + ∗4 − " (on appelle cela une chaîne de *tokens*). Écrire une fonction `calc_exp_arith(ch)` qui prend en paramètre la chaîne `ch` et qui renvoie la valeur numérique de l'expression arithmétique.

## La structure File

**Exercice 5 :**

Une **file** est une structure de données de type **FIFO** (**F**irst **I**n **F**irst **O**ut) : le premier entré est le premier sorti. On suppose qu'on a les primitives suivantes dans le fichier « `file.py` » :

. `Est_vide(f)` : renvoie « True » si la file est vide, « False » sinon.
. `sommet(f)` : renvoie le premier élément de la file.
. `defiler(f)` : supprime de la file le premier élément.
. `enfiler(f,elt)` : ajoute dans la file l'élément elt.

Écrivez les fonctions suivantes :

1. `afficher()` : cette fonction affiche tous les éléments de la file.
2. `defilerJusqua(elt)` : cette méthode défile la file jusqu'à l'élément « elt ». L'élément « elt » n'est pas défilé. Si l'élément n'appartient pas à la file, alors la méthode défile tous les éléments de la file.

**Exercice 6 :**

On pourra éventuellement utiliser une ou des piles/files temporaires, on utilisera les primitives `creer_pile()` qui renvoie une pile vide et `creer_file()` qui renvoie une file vide.
Écrivez les méthodes suivantes.

1. `appartient(p,elt)` : cette fonction s'applique sur une Pile renvoie « True » si l'élément appartient à la pile, « False » sinon. Attention : il est important que la pile ne change pas.
2. `Inverser_file(f)` : cette fonction inverse les éléments de la file à laquelle elle est appliquée. On a le droit d'utiliser des files ou des piles temporaires.
3. `Inverser_pile(p)` : cette fonction inverse les éléments de la pile à laquelle elle est appliquée. On interdit l'utilisation de files. Seules des piles temporaires peuvent être utilisées.