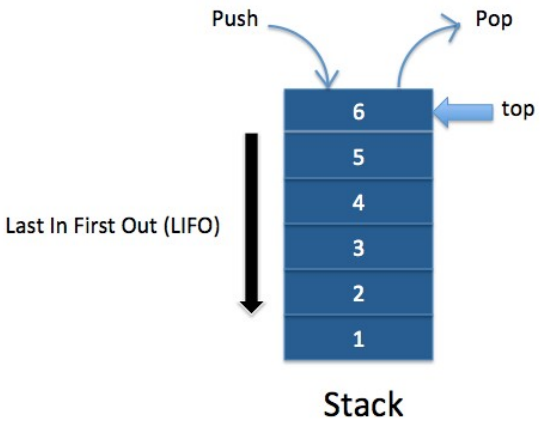
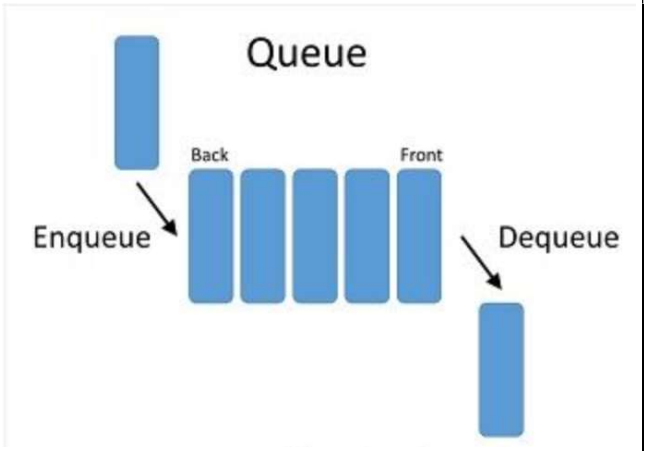
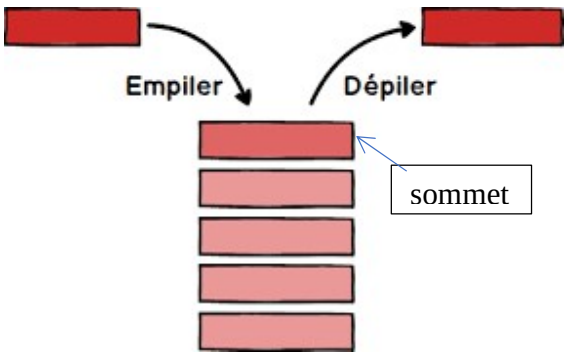
T NSI Structures de données: Piles & Files

1. Définitions

Piles (en anglais stack)	Files (en anglais *Queue*)
	
LIFO : Last In First Out	**FIFO** : First In First Out
	
• Sert à mémoriser les pages Web visitées par un navigateur • Évaluer des expressions mathématiques en notation post-fixée (polonaise inverse) • Gestion de la mémoire par les langage de programmation (pile et tas)	• Mémoriser temporairement des transactions qui doivent attendre pour être traitées • Serveurs d'impression : traiter les requêtes dans l'ordre dans lequel elles arrivent • Moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune

2. Les Piles Mode LIFO (Last in First out)

2.1. Fonctions/méthodes



«empiler» : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».

«dépiler» : 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

2.2. Une implémentation utilisant les listes « pythonesques »

```
class Pile:

    def __init__(self):


    def empiler(self, valeur):


    def depiler(self):
```

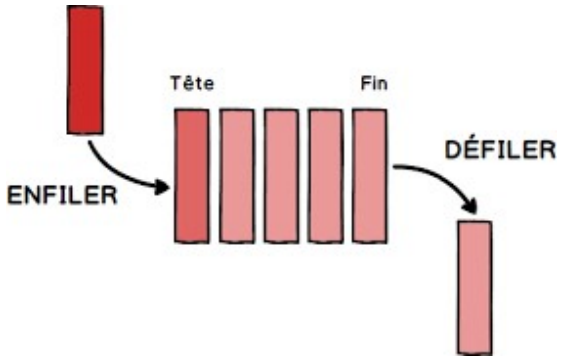
liste.pop() : enlève et renvoie le dernier élément de la liste

```
    def estVide(self):


    def taille(self):


    def __str__(self):
        ch = ''
        for x in self.valeurs:
            ch = "|\t" + str(x) + "\t|" + "\n" + ch
        ch = "\nEtat de la pile:\n" + ch
        return ch
```

```
p = Pile()
p.empiler(9)
p.empiler(2)
p.empiler(5)

print(p)
print("taille pile : ", p.taille())

p.depiler()
p.empiler(7)

print(p.estVide())

print(p)
```

3. Les Files Mode FIFO (First In First Out)

3.1. Fonctions/méthodes

	«enfiler» : ajoute un élément sur la file. Terme anglais correspondant : « enqueue ».
	«défiler» : renvoie le prochain élément de la file, et le retire de la file. Terme anglais : « dequeue »
	«est_Vide» : renvoie vrai si la file est vide, faux sinon
	«taille» : renvoie le nombre d'éléments dans la pile

3.2. Une implémentation utilisant les listes « pythonesques »

```
class File:

    def __init__(self):


    def enfiler(self, valeur):


    def defiler(self):




    def est_Vide(self):




    def taille(self):



    def __str__(self):
        ch = "\nEtat de la file:\n"
        for x in self.valeurs:
            ch +=  str(x) + " "
        return ch
```

```
q = File()
q.enfiler(9)
q.enfiler(2)
q.enfiler(5)

print(q)

q.defiler()
q.enfiler(7)

print("La file est-
elle vide ? : ", q.est_Vide())

print(q)
print("Longueur de la file:", q.taille())
```

4. Utilisation type d'une Pile : Evaluation d'expressions mathématiques :

Méthode d'évaluation d'une expression postfixe (ou encore polonaise inverse):

- on itère sur les éléments de T (itération i) en utilisant une pile P
- si T[i] est un nombre alors on l'empile dans P
- si T[i] est un opérateur alors on évalue l'opération entre les deux premiers éléments de la pile (que l'on dépile), et on empile le résultat
- à la fin de l'itération le résultat est au sommet de la pile

Exemple 1 : x = (a + b) * 5

Rappel sur les priorités des opérateurs arithmétiques
- * et / sont plus prioritaires que + et –
- = est le moins prioritaire
-

Les différentes notations possibles
- Notation infixé : a + 1
- Notation préfixée : + a 1
- Notation postfixée : a 1 +

L'expression de x = (a + b) * 5 en notation postfixée devient : x (a b +) 5 * =

Les parenthèses deviennent inutiles (pas d'ambigüité) d'où : x a b + 5 * =

Remplissez la pile puis dépiler en suivant l'algorithme décrit par la méthode

```
expression = { "x","a","b","+",5,"*","="}
```

temps	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Etat pile										
Résultat										

Exemple 2 : y = ((objet – 1) + x * tmp / 8) / 25 – 12

Devient y ((((objet 1 –) ((x tmp *) 8 /) +) 25 /) 12 -) =

Puis sans parenthèse : y objet 1 – x tmp * 8 / + 25 / 12 - =