# Exercices - Piles / Files

**Exercice 1.** : Des parenthèses et des Piles

On dispose de chaînes de caractères contenant uniquement des parenthèses ouvrantes et fermantes.

Un parenthésage est correct si :

- le nombre de parenthèses ouvrantes de la chaîne est égal au nombre de parenthèses fermantes ;
- en parcourant la chaîne de gauche à droite, le nombre de parenthèses déjà ouvertes doit être, à tout moment, supérieur ou égal au nombre de parenthèses déjà fermées.

Ainsi, ((( )( ))) est un parenthésage correct. Les parenthésages ( ))(() et (( ))(( ) sont, eux, incorrects.

On souhaite programmer une fonction bon_parenthesage() qui prend en paramètre une chaîne de caractères ch formée de parenthèses et renvoie True si la chaîne est bien parenthésée et False sinon. Cette fonction utilise une pile et suit le principe suivant :

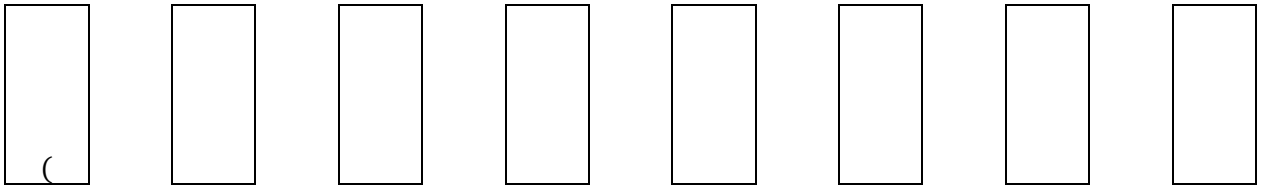
En parcourant la chaîne de gauche à droite, si on trouve une parenthèse ouvrante, on l'empile au sommet de la pile et si on trouve une parenthèse fermante, on dépile (si possible) la parenthèse ouvrante stockée au sommet de la pile.

La chaîne est alors bien parenthésée si, à la fin du parcours, la pile est vide. Elle est, par contre, mal parenthésée :

- si dans le parcours, on trouve une parenthèse fermante, alors que la pile est vide ;
- ou si, à la fin du parcours, la pile n'est pas vide

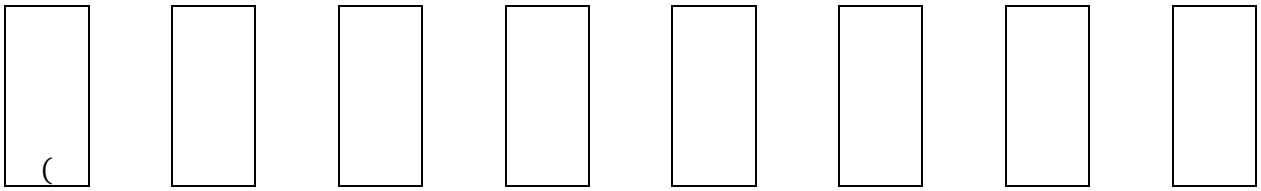
1- Exécution à la main de cet algorithme :

Evolution du contenu de la pile pour vérifier le parenthésage de c = '((( )( ))' :



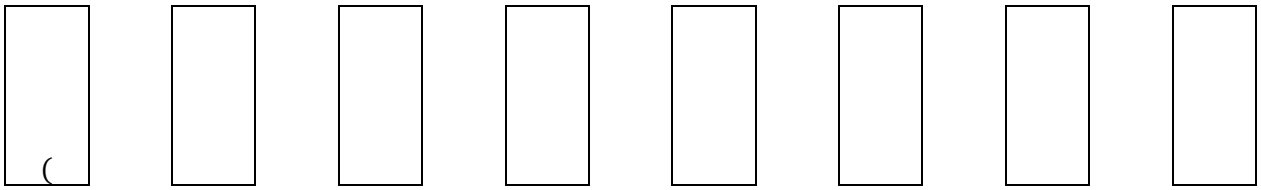

Valeur renvoyée par la fonction :

Evolution du contenu de la pile pour vérifier le parenthésage de c = '( ))(()' :




Valeur renvoyée par la fonction :

Evolution du contenu de la pile pour vérifier le parenthésage de c = '(( ))(( )' :




Valeur renvoyée par la fonction :

2- Implémentation :

Dans un fichier nommé *exercice1.py* , écrire le code de la class Pile des tps précédents et écrire le script de la fonction bon_parenthesage(). On utilisera les méthodes empiler(), estVide() et depiler(). Tester l'ensemble avec les exemples suivants :

```
# Main
if __name__ == '__main__' :
    essais = ['((())))' , '()(()' , '((()(()' ,'(((2+4)*(5-1)))' ]
    for e in essais :
        ret = bon_parenthesage(e)
        print(f"Pour {e} : {ret}")
```

```
>>> (executing file "exercice1.py")
Pour ((())))  : True
Pour ()(()  : False
Pour ((()(()  : False
Pour (((2+4)*(5-1)))  : True
```

```
def bon_parenthesage(ch) :
    p = Pile()
    for c in ch:
        if c == '(':
            p.empiler(c)
        elif c == ')':
            if p.estVide():
                return False
            else:
                p.depiler()
    return p.estVide()
```

CORRIGE

**Exercice 2.** : Des nombres positifs et des Files

Télécharger sur *nsibranly.fr* le fichier exercice2.py qui contient uniquement le code d'une class File.

Compléter ce fichier en écrivant le code **d'une fonction** nommée *positifs()* qui prend une file de nombres entiers en paramètre. Elle renvoie une **nouvelle** file contenant les entiers positifs de la file initiale, dans le même ordre.

Vous n'utiliserez que les méthodes empiler(), estVide() et depiler().

```
# Main
if __name__ == '__main__' :
    from random import randint
    f = File('f')
    for i in range(10) :
        x = randint(-10,10)
        f.enfiler(x)
    print(f)

    F = positifs(f)
    print(F)
    print(f)
```

Tester l'ensemble avec l'exemple dont le programme principal est donné ci-dessus et dont le résultat dans la console est donné ci-contre :

```
>>> (executing file "exercice2.py")

Etat de la file f :

-----------------------------------------------------
 -> --(8)--(3)--(-5)--(9)--(-3)--(8)--(-1)--(7)--(3)--(-6)
-----------------------------------------------------

Etat de la file f positifs :

------------------------------
 -> --(8)--(3)--(9)--(8)--(7)--(3)
------------------------------

Etat de la file f :

-----------------------------------------------------
 -> --(8)--(3)--(-5)--(9)--(-3)--(8)--(-1)--(7)--(3)--(-6)
-----------------------------------------------------
```

CORRIGE

```
def positifs(f) :
    n = f.taille()
    fPositif = File('f positifs')
    for i in range(n) :
        x = f.defiler()
        f.enfiler(x)
        if x >= 0 :
            fPositif.enfiler(x)
    return fPositif
```