ALG. Piles_Files_Exercices. Corrigé

***Exercice 1* : de nouvelles méthodes pour la classe Stack**
Le plus simple est d'utiliser la définition de la classe Stack par le biais de la list en Python.
Pour vider « proprement » une pile, il faut supprimer les éléments un par un ou utiliser la méthode clear().
***Une instruction comme self.data = None n'est donc pas acceptable** même si cela fonctionne en langage Python !*

Rappel : des langages de bas niveau comme le C nécessitent de désallouer la mémoire prise par chaque élément d'un tableau. Autant prendre rapidement les bonnes habitudes.

```
# Définition de la classe de type `Pile`
class Stack :
    def __init__(self):
        self.data = []

    # Si la pile n'a aucun élément alors elle est vide
    def empty(self):
        return len(self.data) == 0

    def clear(self) :
        while not self.empty() :
            # Elément à récupérer et à désallouer dans
            # certains langages
            self.data.pop()

    def size(self) :
        return len(self.data)

    # Ajout d'un élément en fin de pile
    def add_back(self,x):
        self.data.append(x)
```

```
    # Suppression du dernier élément de la pile
    # et renvoi de sa valeur
    def pop_back(self):
        # Gestion du cas d'une pile vide
        if self.empty() == True :
            raise IndexError("La pile est vide")
        else :
            return self.data.pop()

    # Surcharge de la fonction `print()`
    def __str__(self):
        s = "|"
        for k in self.data :
            s = s + str(k) + "|"
        return s


# Jeu de test
myPile = Stack()
myPile.add_back(5)
myPile.add_back("t")
myPile.add_back("yop")
print(myPile.size()) # Attendu : 3
myPile.clear()
print(myPile.size()) # Attendu : 0
print(myPile.empty()) # Attendu : True
```

```
3
0
True
```

***Exercice 2* : séparation d'une pile en deux**

```
def share(myPil) :
    # Il faut inverser l'ordre de la pile
    # passée en paramètre
    invert_stack = Stack()
    pair_stack = Stack() # Pour les nbres pairs
    unpair_stack = Stack() # et les impairs

    # Inversion de l'ordre de `myPil`
    while not myPil.empty() :
        invert_stack.add_back(myPil.pop_back())

    # Parcours de `invert_pil` et tri en fonction
    # de la parité des nombres
    while not invert_stack.empty() :
        # Récupération de la valeur dépilée
        value = invert_stack.pop_back()
        # Si `value` est pair
        if not value % 2 :
            pair_stack.add_back(value)
        else :
            unpair_stack.add_back(value)

    return pair_stack, unpair_stack
```

```
# Jeu de tests
my_stack = Stack()
my_stack.add_back(5)
my_stack.add_back(2)
my_stack.add_back(8)
my_stack.add_back(9)
my_stack.add_back(11)
my_stack.add_back(4)

pair_s , unpair_s = share(my_stack)
print(pair_s)  # Attendu : |2|8|4|
print(unpair_s)# Attendu : |5|9|11|


```

```
|2|8|4|
|5|9|11|
```

Remarque : il faut bien penser à inverser l'ordre des éléments de la pile passée en paramètre (variable *invert_stack* ici).

Rappel : l'instruction *val % 2* renvoie 0 si val est un nombre pair et 1 s'il est impair.

Exercice 3 : vérification du bon parenthésage

1/ 2/

```
def parenthesage(txt) :
    myStack = Stack()

    for char in txt :
        if char == '(' :
            myStack.add_back(1)
        if char == ')' :
            myStack.pop_back()

    return myStack
```

```
# Jeu de tests
print(parenthesage("(()())")) # Attendu : |
print(parenthesage("(()")) # Attendu : |1|
print(parenthesage("())()(")) # Attendu : erreur !

```

```
|
|1|

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-6-aec031656691> in <module>
     45 print(parenthesage("(()())")) # Attendu : |
     46 print(parenthesage("(()")) # Attendu : |1|
---> 47 print(parenthesage("())()(")) # Attendu : erreur !
```

3/ Si la chaîne est bien parenthésée, le programme s'exécute correctement et renvoie une pile vide.

4/

```
def parenthesage(txt) :
    myStack = Stack()

    for char in txt :
        # Ajout d'un élément dans la pile en cas
        # d'une quelconque parenthèse ouvrante
        if char == '(' or char == '[':
            # On ajoute la parenthèse ouvrante
            # correspondante
            myStack.add_back(char)

        elif char == ')' or char == ']' :
            # Si la pile est vide, c'est qu'il manque
            # une parenthèse ouvrante
            if myStack.empty() :
                return False

            # On dépile `myStack`
            value = myStack.pop_back()
            # La valeur dépilée doit être la parenthèse
            # ouvrante correspondante
            if value + str(char) != '()' and value + str(char) != '[]' :
                return False

        # S'il s'agit d'un autre caractère, on passe
        else :
            pass

    return myStack.empty()
```

```
# Jeu de tests
print(parenthesage("(3 + 7)")) # Attendu : True
print(parenthesage("(4[6 - 8])")) # Attendu : True
print(parenthesage("3( -5 [6x + 8) ])")) # Attendu : False !

```

```
True
True
False
```

Remarque : ce type de programme peut être utilisé pour vérifier le bon parenthésage d'une expression littérale mathématique.

<u>Exercice 4</u> : affichage d'une file construite à partir de deux piles

```
# Affichage comme une "queue" des éléments
def __str__(self):
    # Il faut donc afficher d'abord la pile d'entrée
    # inversée puis celle de la sortie

    # Stocke la pile d'entrée inversée
    invert_entry = Stack()
    stack_entry = deepcopy(self.entry)

    while not stack_entry.empty() :
        invert_entry.add_back(stack_entry.pop_back())

    # Et l'affichage de cette pile d'entrée inversée
    s = "|"
    for c in invert_entry.data :
        s += str(c) + "|"

    # Affiche la pile de sortie
    for c in self.exit.data :
        s += str(c) + "|"

    return s
```

```
83 # Jeu de tests
84 myFile = File()
85 myFile.add_back("45")
86 myFile.add_back("a")
87 myFile.add_back(-1)
88 print(myFile) # Attendu |-1|a|45|
89 myFile.pop_front()
90 print(myFile) # Attendu |-1|a|
91 myFile.add_back("yop")
92 myFile.add_back("0")
93 print(myFile) # Attendu [0|yop|-1|a|
94
```

```
|-1|a|45|
|-1|a|
|0|yop|-1|a|
```

<u>Remarque</u> : cette méthode affichera « | » si la file est vide. On peut prévoir une instruction conditionnelle d'interception dans ce cas.