

TD_Pile_File

May 30, 2020

```
In [173]: from TD_Listes import Liste, Cellule
```

1 Classe Pile

Construire une classe implémentant les piles (sans utiliser le module queue de Python), incluant : * un constructeur de paramètres self et une liste L * un attribut liste (du type list classique de Python) * la surcharge de la méthode d'affichage * des méthodes empiler, depiler, sommet, estVide, hauteur et traiter

1.1 Classe Pile avec tableau dynamique en Python

```
In [174]: class Pile:
           """à partir d'un tableau dynamique en Python"""

           def __init__(self, liste):
               self.liste = liste

           def __str__(self):
               output = ''
               for e in reversed(self.liste):
                   output += "|{:~15}|\n".format(e)
                   output += "|{:~15}|\n".format('_ '*15)
               return output

           def __repr__(self):
               repr(self.liste)

           def depiler(self):
               assert not self.est_vide(), "La pile est vide !"
               self.liste.pop()

           def empiler(self, e):
               self.liste.append(e)

           def est_vide(self):
               return self.liste == []
               #return len(self.liste) == 0 #non sinon on utiliserait len pour la hauteur
```

Fonctionnalités de Pile

- **Constructeur** `Pile()`
 - Postconditions : la pile est une pile vide
- **Destructeur** `~Pile()`
 - Postconditions : libération de la mémoire utilisée, la pile est une pile vide
- **Procédure** `empiler (e)`
 - Postcondition : e est ajouté en sommet de la pile
- **Procédure** `dépiler ()`
 - Précondition : la pile n'est pas vide
 - Postcondition : le sommet de la pile est dépilé
- **Procédure** `vider ()`
 - Postcondition : la pile ne contient plus aucun élément
- **Fonction** `estVide () : booléen`
 - Résultat : retourne vrai si la pile est vide, faux sinon
- **Fonction** `sommet () : tout type`
 - Précondition : la pile n'est pas vide
 - Résultat : retourne le sommet de la pile
- **Fonction** `traiter () : tout type`
 - Précondition : la pile n'est pas vide
 - Postcondition : le sommet de la pile est dépilé
 - Résultat : retourne le sommet de la pile

Fonctionnalités d'une pile

```
def sommet(self):
    assert not self.est_vide(), "La pile est vide !"
    return self.liste[-1]

def traiter(self):
    assert not self.est_vide(), "La pile est vide !"
    element = self.sommet()
    #return self.liste.pop()
    self.depiler()
    return element

def hauteur(self):
    pile2 = Pile([])
    compteur = 0
    #on utilise est_vide
    #donc il ne faut pas utiliser hauteur dans est_vide
    while not self.est_vide():
        pile2.empiler(self.traiter())
        compteur += 1
    #on reconstruit la pile
    while not pile2.est_vide():
        self.empiler(pile2.traiter())
```

```

        return compteur

    def vider(self):
        while not self.est_vide():
            self.depiler()

    def detruire(self):
        self.vider()
        del self.liste

```

In [175]: *# Exemple d'utilisation*

```

stack = Pile([])
v = stack.est_vide()
print("Pile vide ? ", v)
stack.empiler(5)
print("On empile 5 : ", stack, sep="\n\n")
stack.empiler(6)
print("On empile 6 : ", stack, sep="\n\n")
stack.empiler(1)
print("On empile 1 : ", stack, sep="\n\n")
print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
stack.depiler()
print("On dépile : ", stack, sep="\n\n")
print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
stack.empiler(7)
print("On empile 7 : ", stack, sep="\n\n")
print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
stack.depiler()
print("On dépile : ", stack, sep="\n\n")
v = stack.est_vide()
print("Pile vide ? ", v)
print("On empile 8 : ", stack, sep="\n\n")
print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
print("Vider la pile")
stack.vider()
print(f"Attributs de la pile : {vars(stack)}")
print("Détruire la pile")
stack.detruire()
print(f"Attributs de la pile : {vars(stack)}")

```

Pile vide ? True

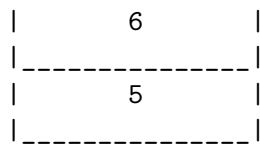
On empile 5 :

```

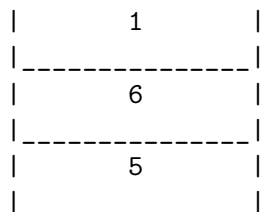
|      5      |
|_-----_|

```

On empile 6 :

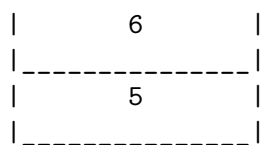


On empile 1 :



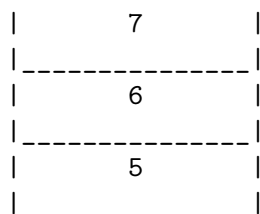
Sommet de la pile 1, hauteur : 3

On dépile :



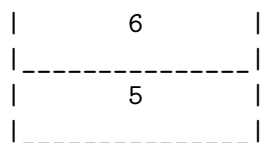
Sommet de la pile 6, hauteur : 2

On empile 7 :



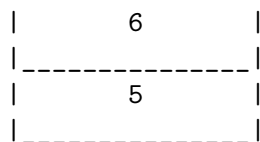
Sommet de la pile 7, hauteur : 3

On dépile :



Pile vide ? False

On empile 8 :



```
Sommet de la pile 6, hauteur : 2
Vider la pile
Attributs de la pile : {'liste': []}
Détruire la pile
Attributs de la pile : {}
```

1.2 Classe Pile avec liste doublement chaînée

```
In [176]: class Pile2(Liste):
           """hérite de la classe Liste de liste doublement chaînée non circulaire"""

           def __init__(self, liste):
               Liste.__init__(self)
               self = Liste.doubleChainedList_from_Pythonlist(liste)

           def __str__(self):
               pointeur = self.premier
               output = ''
               while pointeur is not None:
                   output += "|{:~15}|\n".format(pointeur.info)
                   output += "|{:~15}|\n".format('_ '*15)
                   pointeur = pointeur.suivant
               return output

           def __repr__(self):
               repr(self)

           def depiler(self):
               assert not self.est_vide(), "La pile est vide !"
               self.supprimerTete()

           def empiler(self, e):
               self.ajouterEnTete(e)

           def sommet(self):
               assert not self.est_vide(), "La pile est vide !"
               return self.iemeElement(0)

           def traiter(self):
               assert not self.est_vide(), "La pile est vide !"
               element = self.sommet()
               self.supprimerTete()
               return element
```

```

def hauteur(self):
    return self.nb_elements()

```

```

In [177]: stack = Pile2([])
          print(vars(stack))

```

```

{'premier': None, 'dernier': None, 'taille': 0}

```

```

In [178]: # Exemple d'utilisation
          stack = Pile2([])
          v = stack.est_vide()
          print("Pile vide ? ", v)
          stack.empiler(5)
          print("On empile 5 : ", stack, sep="\n\n")
          stack.empiler(6)
          print("On empile 6 : ", stack, sep="\n\n")
          stack.empiler(1)
          print("On empile 1 : ", stack, sep="\n\n")
          print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
          stack.depiler()
          print("On dépile : ", stack, sep="\n\n")
          print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
          stack.empiler(7)
          print("On empile 7 : ", stack, sep="\n\n")
          print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
          stack.depiler()
          print("On dépile : ", stack, sep="\n\n")
          v = stack.est_vide()
          print("Pile vide ? ", v)
          print("On empile 8 : ", stack, sep="\n\n")
          print(f"Sommet de la pile {stack.sommet()}, hauteur : {stack.hauteur()}")
          print("Vider la pile")
          stack.vider()
          print(f"Attributs de la pile : {vars(stack)}")
          print("Détruire la pile")
          stack.detruire()
          print(f"Attributs de la pile : {vars(stack)}")

```

```

Pile vide ? True

```

```

On empile 5 :

```

```

|      5      |
|_-----_|

```

```

On empile 6 :

```

```

|      6      |

```

5

On empile 1 :

1
6
5

Sommet de la pile 1, hauteur : 3

On dépile :

6
5

Sommet de la pile 6, hauteur : 2

On empile 7 :

7
6
5

Sommet de la pile 7, hauteur : 3

On dépile :

6
5

Pile vide ? False

On empile 8 :

6
5

Sommet de la pile 6, hauteur : 2

Module File

- **Constructeur** `File()`
 - Postconditions : la file est une file vide
- **Destructeur** `~File()`
 - Postconditions : libération de la mémoire utilisée, la file est une file vide
- **Procédure** `enfiler (e)`
 - Postcondition : e est ajouté à la file
- **Procédure** `défiler ()`
 - Précondition : la file n'est pas vide
 - Postcondition : le premier de la file est supprimé
- **Procédure** `vider ()`
 - Postcondition : la file ne contient plus aucun élément
- **Fonction** `estVide ()` : booléen
 - Résultat : retourne vrai si la file est vide, faux sinon
- **Fonction** `premierDeLaFile ()` : tout type
 - Précondition : la file n'est pas vide
 - Résultat : retourne le premier de la file
- **Fonction** `traiter()` : tout type
 - Précondition : la file n'est pas vide
 - Postcondition : le premier de la file est supprimé
 - Résultat : retourne le premier de la file

Fonctionnalités d'une file

Vider la pile

Attributs de la pile : {'premier': None, 'dernier': None, 'taille': 0}

Détruire la pile

Attributs de la pile : {}

1.3 Classe File

Construire une classe implémentant les files (sans utiliser le module queue de Python), incluant : * un constructeur de paramètres self et une liste L * un attribut liste (du type list classique de Python) * la surcharge de la méthode d'affichage * des méthodes traiter, enfiler, estVide et longueur

1.4 Classe File avec tableau dynamique en Python

1.4.1 Implémentation à l'aide d'un tableau dynamique en Python

```
In [179]: class File:
           """à partir d'un tableau dynamique en Python"""

           def __init__(self, liste):
               self.liste = liste

           def __str__(self):
               output = 'Tte : '
               for e in self.liste:
```



```

        output += "{:~15}<--".format(e)
    return output.rstrip('<--') + ': Queue'

def __repr__(self):
    repr(self.liste)

def defiler(self):
    assert not self.est_vide(), "La file est vide !"
    self.liste.pop(0)

def enfiler(self, e):
    self.liste.append(e)

def est_vide(self):
    return self.liste == []
    #return len(self.liste) == 0 #non sinon on utiliserait len pour la hauteur

def premierDeLaFile(self):
    assert not self.est_vide(), "La file est vide !"
    return self.liste[0]

def traiter(self):
    assert not self.est_vide(), "La pile est vide !"
    element = self.premierDeLaFile()
    #return self.liste.pop()
    self.defiler()
    return element

def longueur(self):
    file2 = File([])
    compteur = 0
    #on utilise est_vide
    #donc il ne faut pas utiliser hauteur dans est_vide
    while not self.est_vide():
        file2.enfiler(self.traiter())
        compteur += 1
    #on reconstruit la file
    while not file2.est_vide():
        self.enfiler(file2.traiter())
    return compteur

def vider(self):
    while not self.est_vide():
        self.defiler()

def detruire(self):
    self.vider()

```

```
del self.liste
```

1.5 Implémentation à l'aide d'une liste doublement chaînée

In [180]: *# Exemple d'utilisation*

```
queue = File([])
v = queue.est_vide()
print("File vide ? ", v)
queue.enfiler(5)
print("On enfile 5 : ", queue, sep="\n\n")
queue.enfiler(6)
print("On enfile 6 : ", queue, sep="\n\n")
queue.enfiler(1)
print("On enfile 1 : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
queue.defiler()
print("On défile : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
queue.enfiler(7)
print("On enfile 7 : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
queue.defiler()
print("On défile : ", queue, sep="\n\n")
v = queue.est_vide()
print("File vide ? ", v)
print("On enfile 8 : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
print("Vider la pile")
queue.vider()
print(f"Attributs de la file : {vars(queue)}")
print("Détruire la pile")
queue.detruire()
print(f"Attributs de la file : {vars(queue)}")
```

File vide ? True

On enfile 5 :

Tte : 5 : Queue

On enfile 6 :

Tte : 5 <-- 6 : Queue

On enfile 1 :

Tte : 5 <-- 6 <-- 1 : Queue

Premier de la file 5,longueur: 3

On défile :

Tte : 6 <-- 1 : Queue

Premier de la file 6, longueur: 2
On enfile 7 :

Tte : 6 <-- 1 <-- 7 : Queue
Premier de la file 6, longueur: 3
On défile :

Tte : 1 <-- 7 : Queue
File vide ? False
On enfile 8 :

Tte : 1 <-- 7 : Queue
Premier de la file 1, longueur: 2
Vider la pile
Attributs de la file : {'liste': []}
Détruire la pile
Attributs de la file : {}

```
In [181]: class File2(Liste):
           """à partir d'une liste doublement chaînée, hérite de la classe Liste
           représentant des lites doublement chaînées"""

           def __init__(self, liste):
               Liste.__init__(self)
               self = Liste.doubleChainedList_from_Pythonlist(liste)

           def __str__(self):
               pointeur = self.premier
               output = 'Tête : '
               while pointeur is not None:
                   output += "{:~15}<--".format(pointeur.info)
                   pointeur = pointeur.suivant
               return output.rstrip('<--') + ': Queue'

           def __repr__(self):
               repr(self.liste)

           def defiler(self):
               assert not self.est_vide(), "La file est vide !"
               self.supprimerTete()

           def enfiler(self, e):
               self.ajouterEnQueue(e)

           def premierDeLaFile(self):
               assert not self.est_vide(), "La file est vide !"
```

```

        return self.iemeElement(0)

    def traiter(self):
        assert not self.est_vide(), "La file est vide !"
        element = self.premierDeLaFile()
        #return self.liste.pop()
        self.defiler()
        return element

    def longueur(self):
        return self.nb_elements()

```

In [182]: *# Exemple d'utilisation*

```

queue = File2([])
v = queue.est_vide()
print("File vide ? ", v)
queue.enfiler(5)
print("On enfile 5 : ", queue, sep="\n\n")
queue.enfiler(6)
print("On enfile 6 : ", queue, sep="\n\n")
queue.enfiler(1)
print("On enfile 1 : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
queue.defiler()
print("On défile : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
queue.enfiler(7)
print("On enfile 7 : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
queue.defiler()
print("On défile : ", queue, sep="\n\n")
v = queue.est_vide()
print("File vide ? ", v)
print("On enfile 8 : ", queue, sep="\n\n")
print(f"Premier de la file {queue.premierDeLaFile()},longueur: {queue.longueur()}")
print("Vider la pile")
queue.vider()
print(f"Attributs de la file : {vars(queue)}")
print("Détruire la pile")
queue.detruire()
print(f"Attributs de la file : {vars(queue)}")

```

File vide ? True

On enfile 5 :

Tête : 5 : Queue

On enfile 6 :

```

Tête :      5      <--      6      : Queue
On enfile 1 :

Tête :      5      <--      6      <--      1      : Queue
Premier de la file 5,longueur: 3
On défile  :

Tête :      6      <--      1      : Queue
Premier de la file 6,longueur: 2
On enfile 7 :

Tête :      6      <--      1      <--      7      : Queue
Premier de la file 6,longueur: 3
On défile  :

Tête :      1      <--      7      : Queue
File vide ? False
On enfile 8 :

Tête :      1      <--      7      : Queue
Premier de la file 1,longueur: 2
Vider la pile
Attributs de la file : {'premier': None, 'dernier': None, 'taille': 0}
Détruire la pile
Attributs de la file : {}

```

1.6 Inversion d'une File en utilisant une Pile

Le but de cet exercice est d'écrire en Python une procédure qui inverse une file d'éléments qui lui est passée en paramètre.

On demande de ne pas utiliser de tableau ou de liste de travail pour effectuer l'inversion, mais d'utiliser plutôt une pile. Il existe en effet une méthode très simple pour inverser une file en utilisant une pile.

```

In [183]: def inverserFile(queue):
            stack = Pile([])
            while not queue.est_vide():
                stack.empiler(queue.traiter())
            while not stack.est_vide():
                queue.enfiler(stack.traiter())

In [184]: queue = File(list(range(5)))
            print("File initiale : \n", queue)
            inverserFile(queue)
            print("File inversée : \n", queue)

```

File initiale :								
Tte :	0	<--	1	<--	2	<--	3	<--
File inversée :								
Tte :	4	<--	3	<--	2	<--	1	<--
								0

1.7 Validité du parenthésage d'une expression

Un problème fréquent pour les compilateurs et les traitements de textes est de déterminer si les parenthèses d'une chaîne de caractères sont équilibrées et proprement incluses les unes dans les autres. On désire donc écrire une fonction qui teste la validité du parenthésage d'une expression :

- on considère que les expressions suivantes sont valides : "()", "[([bonjour+]essai)7plus-];"
- alors que les suivantes ne le sont pas : "(", ")(", "4(essai)".

Notre but est donc d'évaluer la validité d'une expression en ne considérant que ses parenthèses et ses crochets. On suppose que l'expression à tester est dans une chaîne de caractères, dont on peut ignorer tous les caractères autres que '(', '[', ']' et ')'. Écrire en Python la fonction valide qui renvoie vrai si l'expression passée en paramètre est valide, faux sinon.

```
In [185]: def parenthesage_valide(expression):
    target = '()[]'
    stack = Pile([])
    fermante = {'(': ')', '[' : ']'}
    ouvrante = { valeur : clef for (clef, valeur) in fermante.items()}
    #construction de la pile de parenthèses/crochets
    for token in expression:
        if token in target:
            stack.empiler(token)
    #dépilement
    stack2 = Pile([])
    while not stack.est_vide():
        token = stack.traiter()
        if token in fermante:
            stack2.empiler(token)
        elif stack2.est_vide():
            return False
        elif stack2.sommet() == ouvrante[token]:
            stack2.depiler()
        else:
            return False
    return stack2.est_vide()
```

```
In [186]: parenthesage_valide("()")
```

```
Out[186]: True
```

```
In [187]: parenthesage_valide(")(")
```

```
Out[187]: False
```

```
In [188]: parenthesage_valide("[]")
Out[188]: False

In [189]: parenthesage_valide("([([bonjour+]essai)7plus- ]);")
Out[189]: True

In [190]: parenthesage_valide("(")
Out[190]: False

In [191]: parenthesage_valide("4(essai)")
Out[191]: False
```