Exercice type bac : Piles et POO — Corrigé

On crée une classe `Pile` qui modélise la structure d'une pile d'entiers.

Le constructeur de la classe initialise une pile vide.

~~La définition de cette classe sans l'implémentation de ses méthodes est donnée ci-dessous.~~

On donne ci-dessous la définition de cette classe et une implémentation fortement inspirée de celle du cours.

```
class Chainon:
    def __init__(self, element=None, suivant=None):
        """element est la valeur du chainon et suivant est le chainon qui suit"""
        self.element = element
        self.suivant = suivant


class Pile:
    def __init__(self):
        """Initialise la pile comme une pile vide"""
        self.summit = Chainon()

    def est_vide(self):
        """Renvoie True si la pile est vide et False sinon"""
        return self.summit.element is None

    def empiler(self, e):
        """Empile l'élément e sur le sommet de la pile, ne renvoie rien"""
        self.summit = Chainon(e, self.summit)

    def depiler(self):
        """Retire l'élément situé au sommet de la pile et le renvoie"""
        item = self.summit.element
        self.summit = self.summit.suivant
        return item

    def nb_elements(self):
        """Renvoie le nombre d'éléments de la pile"""
        long = 0
        chainon = self.summit
        while chainon.element is not None:
            chainon = chainon.suivant
            long = long + 1
        return long

    def afficher(self):
        """Affiche de gauche à droite les éléments de la pile,..."""
        temp = Pile()
        while not self.est_vide():
            temp.empiler(self.depiler())
        res = ""
        while not temp.est_vide():
            item = temp.depiler()
```

```
            res = res + str(item) + ", "
            self.empiler(item)
        if res == "":
            print("pile vide")
        else:
            print(res[:-2])
```

Seules les méthodes de la classe ci-dessus doivent être utilisées pour manipuler les objets `Pile`.

1. a. Écrire une suite d'instructions permettant de créer une instance de la classe `Pile` affectée à une variable `pile1` contenant les éléments 7, 5 et 2 insérés dans cet ordre.

Ainsi, à l'issue de ces instructions, l'instruction `pile1.afficher()` produit l'affichage : `a, 5, 2`.

Réponse :

```
pile1 = Pile()
pile1.empiler(7)
pile1.empiler(5)
pile1.empiler(2)
pile1.afficher() # pour vérifier
```

```
7, 5, 2
```

1. b. Donner l'affichage produit après l'exécution des instructions suivantes.

```
element1 = pile1.depiler() # pile 1 est maintenant composée de 7 et 5 ; element1 vaut 2
pile1.empiler(5) # pile 1 est maintenant composée de 7, 5 et 5
pile1.empiler(element1) # pile 1 est maintenant composée de 7, 5, 5 et 2
pile1.afficher()
```

Réponse :

```
7, 5, 5, 2
```

2. On donne la fonction `mystere` suivante :

```
def mystere(pile, element):
    pile2 = Pile()
    nb_elements = pile.nb_elements()
    for i in range(nb_elements):
        elem = pile.depiler()
        pile2.empiler(elem)
        if elem == element:
            return pile2
    return pile2
```

2. a. Dans chacun des quatre cas suivants, quel est l'affichage obtenu dans la console ?

```
# Cas n°1
pile = Pile() # construction de la pile
pile.empiler(7)
pile.empiler(5)
pile.empiler(2)
pile.empiler(3)
pile.afficher() # vérification
mystere(pile, 2).afficher() # réponse
```

Réponse :

```
7, 5, 2, 3
3, 2
```

```
# Cas n°2
pile = Pile() # construction de la pile
pile.empiler(7)
pile.empiler(5)
pile.empiler(2)
pile.empiler(3)
pile.afficher() # vérification
mystere(pile, 9).afficher() # réponse
```

Réponse :

```
7, 5, 2, 3
3, 2, 5, 7
```

```
# Cas n°3
pile = Pile() # construction de la pile
pile.empiler(7)
pile.empiler(5)
pile.empiler(2)
pile.empiler(3)
pile.afficher() # vérification
mystere(pile, 3).afficher() # réponse
```

Réponse :

```
7, 5, 2, 3
3
```

```
# Cas n°4
pile = Pile() # construction de la pile
print(pile.est_vide()) # vérification
mystere(pile, 3).afficher() # réponse
```

Réponse :

```
True
pile vide
```

2. b. Expliquer ce que permet d'obtenir la fonction `mystere`.

Réponse :

La fonction mystère renvoie une pile constituée des éléments de la pile donnée en paramètre, en partant de son sommet et en s'arrêtant lorsqu'un élément égal au paramètre `element` est trouvé. En particulier, dans le cas où `element` n'est pas dans la pile, la fonction retourne la pile inversée.

3. Écrire une fonction `etendre(pile1, pile2)` qui prend en arguments deux objets `Pile` appelés `pile1` et `pile2` et qui modifie `pile1` en lui ajoutant les éléments de `pile2` rangés dans l'ordre inverse. Cette fonction ne renvoie rien.

On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>> pile1.afficher()
7, 5, 2, 3
>>> pile2.afficher()
1, 3, 4
>>> etendre(pile1, pile2)
>>> pile1.afficher()
7, 5, 2, 3, 4, 3, 1
>>> pile2.est_vide()
True
```

Réponse : Code le la fonction `etendre` :

```
def etendre(pile1, pile2):
    """modifie `pile1` en lui ajoutant les éléments de `pile2` rangés dans l'ordre inverse"""
    while not pile2.est_vide():
        pile1.empiler(pile2.depiler())

# test
pile1 = Pile() # construction de pile1
pile1.empiler(7)
pile1.empiler(5)
pile1.empiler(2)
pile1.empiler(3)
pile2 = Pile() # construction de pile2
pile2.empiler(1)
pile2.empiler(3)
pile2.empiler(4)
pile1.afficher()
pile2.afficher()
etendre(pile1, pile2)
pile1.afficher()
pile2.est_vide()
```

Sortie :

```
7, 5, 2, 3
1, 3, 4
7, 5, 2, 3, 4, 3, 1


True
```

4. Écrire une fonction `supprime_toutes_occurences(pile, element)` qui prend en argument un objet `Pile` appelé `pile` et un élément `element` et supprime tous les éléments `element` de `pile`.

On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>> pile.afficher()
7, 5, 2, 3, 5
>>> supprime_toutes_occurences(pile, 5)
>>> pile.afficher()
7, 2, 3
```

Réponse : La fonction demandée peut s'écrire :

```
def supprime_toutes_occurences(pile, element):
    """supprime tous les éléments `element` de `pile`"""
    temp = Pile() # pile accessoire pour vider le pile
    while not pile.est_vide(): # on depile un à un les éléments de pile
        item = pile.depiler()
        if item != element:
            temp.empiler(item) # on les empile dans temp s'ils sont différents de element
    while not temp.est_vide():
        pile.empiler(temp.depiler()) # on rempile tout dans pile en dépilant temp

# Vérification
pile = Pile() # construction de pile1
pile.empiler(7)
pile.empiler(5)
pile.empiler(2)
pile.empiler(3)
pile.empiler(5)
pile.afficher()
supprime_toutes_occurences(pile, 5)
pile.afficher()
```

Sortie :

```
7, 5, 2, 3, 5
7, 2, 3
```