

Contrôle de connaissances

Nom : Prénom :

Exercice 1 : (6 points)

Compléter les méthodes pour la construction de la classe `Pile`

Classe `Pile`

```
class Pile:
    def __init__(self):
        self.contenu = []

    def emmpile(self, valeur):
        .
        .
        .
        .

    def pile_vide(self):
        .
        .
        .
        .

    def depile(self):
        .
        .
        .
        .
        .
```

Exercice 2 : (6 points)

Seules les méthodes de l'exercice précédent pourront être utilisée dans les questions suivantes.
:

1. Compléter la fonction `nombre_de_terme` qui prend une pile, et renvoie le nombre d'éléments de la pile. A la fin de l'utilisation de la fonction, la pile ne doit pas être changée :

```
def nombre_de_terme (pile) :  
    nb = ..... # le compteur  
    pile_2 = Pile()  
    while not( ..... )= :  
        tete =  
        nb =  
        pile_2.empile(.....)  
  
    while not ( ..... )= :  
        pile.empile(.....)  
  
    return .....
```

2. Compléter la fonction `sans_zero` qui prend une pile, et élimine les 0 éventuelles sans changer l'ordre des éléments :

```
def sans_zero(pile) :  
    pile_2 = .....  
    while not(.....) :  
        tete =.....  
        if tete != 0 :  
            pile_2.....  
  
    while not (.....) :  
        .....
```

3. Compléter la fonction `separe` qui prend une pile, et renvoie deux piles : une pile d'éléments positifs et une pile d'éléments négatifs :

```
def separe (pile) :  
    pile_positif = .....  
  
    pile_negatif = .....  
  
    while ..... :  
        tete =  
        if tete > 0 :  
            .....  
  
        else :  
            .....  
  
    return ( pile_positif , pile_negatif )
```

Exercice 3 : (8 points)

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.

```
class Pile:
    def __init__(self):
        """Initialise la pile comme une pile vide."""
    def est_vide(self):
        """Renvoie True si la liste est vide, False sinon."""
    def empiler(self, e):
        """Ajoute l'element e sur le sommet de la pile,
        ne renvoie rien."""
    def depiler(self):
        """Retire l'element au sommet de la pile et le renvoie."""
    def nb_elements(self):
        """Renvoie le nombre d'elements de la pile. """
    def afficher(self):
        """Affiche de gauche a droite les elements de la pile, du fond
        de la pile vers son sommet. Le sommet est alors l'element
        affiche le plus a droite. Les elements sont separes par une
        virgule. Si la pile est vide la methode affiche 'pile
        vide'."""
```

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 : 7,5,2.

.....

.....

.....

.....

.....

- (b) Donner l'affichage produit après l'exécution des instructions suivantes.

```
element1 = pile1.depiler()
pile1.empiler(5)
pile1.empiler(element1)
pile1.afficher()
```

.....

.....

.....

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
```

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

- Cas n°1 :

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

.....
.....

- Cas n°2 :

```
>>>pile.afficher()  
7, 5, 2, 3  
>>>mystere(pile, 9).afficher()
```

.....
.....

- Cas n°3 :

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

.....
.....

- Cas n°4 :

```
>>>pile.est_vide()  
True  
>>>mystere(pile, 3).afficher()
```

.....
.....

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

.....
.....
.....
.....
.....

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
```

.....

.....

.....

.....

.....

4. Écrire une fonction `supprime_toutes_occurences(pile,element)` qui prend en arguments 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
```

.....

.....

.....

.....

.....